import sys

import os

sys.path.append(os.getcwd())

from proj.inventory import MobileInventory, InsufficientException

import pytest

import unittest

# Import MobileInventory class and InsufficientException from the inventory module using the expression from proj.inventory import MobileInventory, InsufficientException.

# Import pytest using the expression import pytest.

# Use assert statement for assert, and to check. Ex: assert 1 == 1

# Define a pytest test class \*\*'TestingInventoryCreation'\*\*

class TestingInventoryCreation:

# Define a pytest test method \*\*'test\_creating\_empty\_inventory'\*\*, which creates an empty inventory and checks if its 'balance\_inventory' is an empty dict using assert.

def test\_creating\_empty\_inventory(self):

c1 = MobileInventory()

assert c1.balance\_inventory == {}

# Define a pytest test method \*\*'test\_creating\_specified\_inventory'\*\*, which checks if inventory instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25}.

def test\_creating\_specified\_inventory(self):

c1 = MobileInventory({'iPhone Model X': 100,'Xiaomi Model Y': 1000,'Nokia Model Z': 25})

assert c1.balance\_inventory == {'iPhone Model X': 100,'Xiaomi Model Y': 1000, 'Nokia Model Z': 25}

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_list'\*\*, which checks if the method raises a TypeError with the message "Input inventory must be a dictionary" when a list "['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z']" is passed as input using assert.

def test\_creating\_inventory\_with\_list(self):

with pytest.raises(TypeError):

c1 = MobileInventory(['iPhone Model X', 'Xiaomi Model Y','Nokia Model Z'])

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_numeric\_keys'\*\*, which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert, when the dict {1:'iPhone Model X', 2:'Xiaomi Model Y', 3:'Nokia Model Z'} is passed as input.

def test\_creating\_inventory\_with\_numeric\_keys(self):

inv\_num = {1:'iPhone Model X',2:'Xiaomi Model Y', 3:'Nokia Model Z'}

with pytest.raises(ValueError,match="Mobile model name must be a string"):

c1 = MobileInventory(inv\_num)

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_nonnumeric\_values'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':'100', 'Xiaomi Model Y': '1000', 'Nokia Model Z':'25'} is passed as input.

def test\_creating\_inventory\_with\_nonnumeric\_values(self):

inventory\_dict = {'iPhone Model X': '100','Xiaomi Model Y': '1000', 'Nokia Model Z':'25'}

with pytest.raises(ValueError,match="No. of mobiles must be a positive integer"):

c1 = MobileInventory(inventory\_dict)

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_negative\_value'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25} is passed as input.

def test\_creating\_inventory\_with\_negative\_value(self):

with pytest.raises(ValueError) :

c1 = MobileInventory({'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25})

# Define another pytest test class \*\*'TestInventoryAddStock'\*\*, which tests the behavior of the \*\*'add\_stock'\*\* method, with the following tests

class TestInventoryAddStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25} and assign it to class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory({'iPhone Model X': 100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25})

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_dict'\*\*, which adds the new stock {'iPhone Model X':50, 'Xiaomi Model Y': 2000, 'Nokia Model A':10} to the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also, check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model X':150, 'Xiaomi Model Y': 3000, 'Nokia Model Z':25, 'Nokia Model A':10} using assert.

def test\_add\_new\_stock\_as\_dict(self):

self.inventory.add\_stock({'iPhone Model X': 50, 'Xiaomi Model Y': 2000, 'Nokia Model A': 10})

assert self.inventory.balance\_inventory == {'iPhone Model X': 150, 'Xiaomi Model Y': 3000, 'Nokia Model Z': 25, 'Nokia Model A': 10}

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_list'\*\*, which adds the new stock ['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'] to the existing inventory, and which checks if the method raises a TypeError with the message "Input stock must be a dictionary" using assert.

def test\_add\_new\_stock\_as\_list(self):

with pytest.raises(TypeError) :

MobileInventory.add\_stock(['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'])

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_numeric\_keys'\*\*, which adds the new stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} to the existing inventory, and which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert.

def test\_add\_new\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_nonnumeric\_values'\*\*, which adds the new stock {'iPhone Model A':'50', 'Xiaomi Model B':'2000', 'Nokia Model C':'25'} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_float\_values'\*\*, which adds the new stock {'iPhone Model A':50.5, 'Xiaomi Model B':2000.3, 'Nokia Model C':25} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_float\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define another pytest test class \*\*'TestInventorySellStock'\*\*, which tests the behavior of the \*\*'sell\_stock'\*\* method, with the following tests

class TestInventorySellStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with the input {'iPhone Model A':50, 'Xiaomi Model B': 2000, 'Nokia Model C':10, 'Sony Model D':1}, and assign it to the class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory(

{'iPhone Model A': 50, 'Xiaomi Model B': 2000, 'Nokia Model C': 10, 'Sony Model D': 1})

# Define a pytest test method \*\*'test\_sell\_stock\_as\_dict'\*\*, which sells the requested stock {'iPhone Model A':2, 'Xiaomi Model B':20, 'Sony Model D':1} from the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model A':48, 'Xiaomi Model B': 1980, 'Nokia Model C':10, 'Sony Model D':0} using assert.

def test\_sell\_stock\_as\_dict(self):

self.inventory.sell\_stock({'iPhone Model A': 2, 'Xiaomi Model B': 20, 'Sony Model D': 1})

assert self.inventory.balance\_inventory == {'iPhone Model A': 48, 'Xiaomi Model B': 1980, 'Nokia Model C': 10,

'Sony Model D': 0}

# Define a pytest test method \*\*'test\_sell\_stock\_as\_list'\*\*, which tries selling the requested stock ['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'] from the existing inventory, and which checks if the method raises a TypeError with the message "Requested stock must be a dictionary" using assert.

def test\_sell\_stock\_as\_list(self):

with pytest.raises(TypeError):

MobileInventory.sell\_stock(['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'])

# Define a pytest test method \*\*'test\_sell\_stock\_with\_numeric\_keys'\*\*, which tries selling the requested stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} from the existing inventory, and which checks if the method raises ValueError with the message "Mobile model name must be a string" using assert.

def test\_sell\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_with\_nonnumeric\_values'\*\*, which tries selling the requested stock {'iPhone Model A':'2', 'Xiaomi Model B':'3', 'Nokia Model C':'4'} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_with\_float\_values'\*\*, which tries selling the requested stock {'iPhone Model A':2.5, 'Xiaomi Model B':3.1, 'Nokia Model C':4} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_float\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_of\_nonexisting\_model'\*\*, which tries selling the requested stock {'iPhone Model B':2, 'Xiaomi Model B':5} from the existing inventory, and which checks if the method raises an InsufficientException with the message "No Stock. New Model Request" using assert.

def test\_sell\_stock\_of\_nonexisting\_model(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_of\_insufficient\_stock'\*\*, which tries selling the requested stock {'iPhone Model A':2, 'Xiaomi Model B':5, 'Nokia Model C': 15} from the existing inventory, and which checks if the method raises an InsufficientException with the message "Insufficient Stock" using assert.

def test\_sell\_stock\_of\_insufficient\_stock(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

8 test pass

import sys

import os

sys.path.append(os.getcwd())

from proj.inventory import MobileInventory, InsufficientException

import pytest

# Import MobileInventory class and InsufficientException from the inventory module using the expression from proj.inventory import MobileInventory, InsufficientException.

# Import pytest using the expression import pytest.

# Use assert statement for assert, and to check. Ex: assert 1 == 1

# Define a pytest test class \*\*'TestingInventoryCreation'\*\*

class TestingInventoryCreation:

# Define a pytest test method \*\*'test\_creating\_empty\_inventory'\*\*, which creates an empty inventory and checks if its 'balance\_inventory' is an empty dict using assert.

def test\_creating\_empty\_inventory(self):

c1 = MobileInventory()

assert c1.balance\_inventory == {}

# Define a pytest test method \*\*'test\_creating\_specified\_inventory'\*\*, which checks if inventory instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25}.

def test\_creating\_specified\_inventory(self):

c2 = MobileInventory({'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25})

assert c2.balance\_inventory == {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25}

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_list'\*\*, which checks if the method raises a TypeError with the message "Input inventory must be a dictionary" when a list "['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z']" is passed as input using assert.

def test\_creating\_inventory\_with\_list(self):

with pytest.raises(TypeError) :

c3 = MobileInventory(['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'])

assert "Input inventory must be a dictionary" == c3.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_numeric\_keys'\*\*, which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert, when the dict {1:'iPhone Model X', 2:'Xiaomi Model Y', 3:'Nokia Model Z'} is passed as input.

def test\_creating\_inventory\_with\_numeric\_keys(self):

with pytest.raises(ValueError):

c4 = MobileInventory({1:'iPhone Model X', 2:'Xiaomi Model Y', 3:'Nokia Model Z'})

assert "Mobile model name must be a string" == c4.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_nonnumeric\_values'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':'100', 'Xiaomi Model Y': '1000', 'Nokia Model Z':'25'} is passed as input.

def test\_creating\_inventory\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError):

c5 = MobileInventory({'iPhone Model X':'100', 'Xiaomi Model Y': '1000', 'Nokia Model Z':'25'})

assert "No. of mobiles must be a positive integer" == c5.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_negative\_value'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25} is passed as input.

def test\_creating\_inventory\_with\_negative\_value(self):

with pytest.raises(ValueError) :

c6 = MobileInventory({'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25})

# Define another pytest test class \*\*'TestInventoryAddStock'\*\*, which tests the behavior of the \*\*'add\_stock'\*\* method, with the following tests

class TestInventoryAddStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25} and assign it to class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory({'iPhone Model X': 100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25})

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_dict'\*\*, which adds the new stock {'iPhone Model X':50, 'Xiaomi Model Y': 2000, 'Nokia Model A':10} to the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also, check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model X':150, 'Xiaomi Model Y': 3000, 'Nokia Model Z':25, 'Nokia Model A':10} using assert.

def test\_add\_new\_stock\_as\_dict(self):

self.inventory.add\_stock({'iPhone Model X': 50, 'Xiaomi Model Y': 2000, 'Nokia Model A': 10})

assert self.inventory.balance\_inventory == {'iPhone Model X': 150, 'Xiaomi Model Y': 3000, 'Nokia Model Z': 25, 'Nokia Model A': 10}

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_list'\*\*, which adds the new stock ['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'] to the existing inventory, and which checks if the method raises a TypeError with the message "Input stock must be a dictionary" using assert.

def test\_add\_new\_stock\_as\_list(self):

with pytest.raises(TypeError) :

MobileInventory.add\_stock(['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'])

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_numeric\_keys'\*\*, which adds the new stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} to the existing inventory, and which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert.

def test\_add\_new\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_nonnumeric\_values'\*\*, which adds the new stock {'iPhone Model A':'50', 'Xiaomi Model B':'2000', 'Nokia Model C':'25'} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_float\_values'\*\*, which adds the new stock {'iPhone Model A':50.5, 'Xiaomi Model B':2000.3, 'Nokia Model C':25} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_float\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define another pytest test class \*\*'TestInventorySellStock'\*\*, which tests the behavior of the \*\*'sell\_stock'\*\* method, with the following tests

class TestInventorySellStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with the input {'iPhone Model A':50, 'Xiaomi Model B': 2000, 'Nokia Model C':10, 'Sony Model D':1}, and assign it to the class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory(

{'iPhone Model A': 50, 'Xiaomi Model B': 2000, 'Nokia Model C': 10, 'Sony Model D': 1})

# Define a pytest test method \*\*'test\_sell\_stock\_as\_dict'\*\*, which sells the requested stock {'iPhone Model A':2, 'Xiaomi Model B':20, 'Sony Model D':1} from the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model A':48, 'Xiaomi Model B': 1980, 'Nokia Model C':10, 'Sony Model D':0} using assert.

def test\_sell\_stock\_as\_dict(self):

self.inventory.sell\_stock({'iPhone Model A': 2, 'Xiaomi Model B': 20, 'Sony Model D': 1})

assert self.inventory.balance\_inventory == {'iPhone Model A': 48, 'Xiaomi Model B': 1980, 'Nokia Model C': 10,

'Sony Model D': 0}

# Define a pytest test method \*\*'test\_sell\_stock\_as\_list'\*\*, which tries selling the requested stock ['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'] from the existing inventory, and which checks if the method raises a TypeError with the message "Requested stock must be a dictionary" using assert.

def test\_sell\_stock\_as\_list(self):

with pytest.raises(TypeError):

MobileInventory.sell\_stock(['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'])

# Define a pytest test method \*\*'test\_sell\_stock\_with\_numeric\_keys'\*\*, which tries selling the requested stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} from the existing inventory, and which checks if the method raises ValueError with the message "Mobile model name must be a string" using assert.

def test\_sell\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_with\_nonnumeric\_values'\*\*, which tries selling the requested stock {'iPhone Model A':'2', 'Xiaomi Model B':'3', 'Nokia Model C':'4'} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_with\_float\_values'\*\*, which tries selling the requested stock {'iPhone Model A':2.5, 'Xiaomi Model B':3.1, 'Nokia Model C':4} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_float\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_of\_nonexisting\_model'\*\*, which tries selling the requested stock {'iPhone Model B':2, 'Xiaomi Model B':5} from the existing inventory, and which checks if the method raises an InsufficientException with the message "No Stock. New Model Request" using assert.

def test\_sell\_stock\_of\_nonexisting\_model(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_of\_insufficient\_stock'\*\*, which tries selling the requested stock {'iPhone Model A':2, 'Xiaomi Model B':5, 'Nokia Model C': 15} from the existing inventory, and which checks if the method raises an InsufficientException with the message "Insufficient Stock" using assert.

def test\_sell\_stock\_of\_insufficient\_stock(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

11 tests pass

import sys

import os

sys.path.append(os.getcwd())

from proj.inventory import MobileInventory, InsufficientException

import pytest

# Import MobileInventory class and InsufficientException from the inventory module using the expression from proj.inventory import MobileInventory, InsufficientException.

# Import pytest using the expression import pytest.

# Use assert statement for assert, and to check. Ex: assert 1 == 1

# Define a pytest test class \*\*'TestingInventoryCreation'\*\*

class TestingInventoryCreation:

# Define a pytest test method \*\*'test\_creating\_empty\_inventory'\*\*, which creates an empty inventory and checks if its 'balance\_inventory' is an empty dict using assert.

def test\_creating\_empty\_inventory(self):

c1 = MobileInventory()

assert c1.balance\_inventory == {}

# Define a pytest test method \*\*'test\_creating\_specified\_inventory'\*\*, which checks if inventory instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25}.

def test\_creating\_specified\_inventory(self):

c2 = MobileInventory({'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25})

assert c2.balance\_inventory == {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25}

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_list'\*\*, which checks if the method raises a TypeError with the message "Input inventory must be a dictionary" when a list "['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z']" is passed as input using assert.

def test\_creating\_inventory\_with\_list(self):

with pytest.raises(TypeError) :

c3 = MobileInventory(['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'])

assert "Input inventory must be a dictionary" == c3.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_numeric\_keys'\*\*, which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert, when the dict {1:'iPhone Model X', 2:'Xiaomi Model Y', 3:'Nokia Model Z'} is passed as input.

def test\_creating\_inventory\_with\_numeric\_keys(self):

with pytest.raises(ValueError):

c4 = MobileInventory({1:'iPhone Model X', 2:'Xiaomi Model Y', 3:'Nokia Model Z'})

assert "Mobile model name must be a string" == c4.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_nonnumeric\_values'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':'100', 'Xiaomi Model Y': '1000', 'Nokia Model Z':'25'} is passed as input.

def test\_creating\_inventory\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError):

c5 = MobileInventory({'iPhone Model X':'100', 'Xiaomi Model Y': '1000', 'Nokia Model Z':'25'})

assert "No. of mobiles must be a positive integer" == c5.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_negative\_value'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25} is passed as input.

def test\_creating\_inventory\_with\_negative\_value(self):

with pytest.raises(ValueError) :

c6 = MobileInventory({'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25})

assert "No. of mobiles must be a positive integer" == c6.balance\_inventory

# Define another pytest test class \*\*'TestInventoryAddStock'\*\*, which tests the behavior of the \*\*'add\_stock'\*\* method, with the following tests

class TestInventoryAddStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25} and assign it to class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory({'iPhone Model X': 100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25})

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_dict'\*\*, which adds the new stock {'iPhone Model X':50, 'Xiaomi Model Y': 2000, 'Nokia Model A':10} to the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also, check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model X':150, 'Xiaomi Model Y': 3000, 'Nokia Model Z':25, 'Nokia Model A':10} using assert.

def test\_add\_new\_stock\_as\_dict(self):

self.inventory.add\_stock({'iPhone Model X': 50, 'Xiaomi Model Y': 2000, 'Nokia Model A': 10})

assert self.inventory.balance\_inventory == {'iPhone Model X': 150, 'Xiaomi Model Y': 3000, 'Nokia Model Z': 25, 'Nokia Model A': 10}

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_list'\*\*, which adds the new stock ['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'] to the existing inventory, and which checks if the method raises a TypeError with the message "Input stock must be a dictionary" using assert.

def test\_add\_new\_stock\_as\_list(self):

with pytest.raises(TypeError) :

MobileInventory.add\_stock(['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'])

assert "Input stock must be a dictionary" == balance\_inventory.add\_stock()

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_numeric\_keys'\*\*, which adds the new stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} to the existing inventory, and which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert.

def test\_add\_new\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_nonnumeric\_values'\*\*, which adds the new stock {'iPhone Model A':'50', 'Xiaomi Model B':'2000', 'Nokia Model C':'25'} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_float\_values'\*\*, which adds the new stock {'iPhone Model A':50.5, 'Xiaomi Model B':2000.3, 'Nokia Model C':25} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_float\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define another pytest test class \*\*'TestInventorySellStock'\*\*, which tests the behavior of the \*\*'sell\_stock'\*\* method, with the following tests

class TestInventorySellStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with the input {'iPhone Model A':50, 'Xiaomi Model B': 2000, 'Nokia Model C':10, 'Sony Model D':1}, and assign it to the class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory(

{'iPhone Model A': 50, 'Xiaomi Model B': 2000, 'Nokia Model C': 10, 'Sony Model D': 1})

# Define a pytest test method \*\*'test\_sell\_stock\_as\_dict'\*\*, which sells the requested stock {'iPhone Model A':2, 'Xiaomi Model B':20, 'Sony Model D':1} from the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model A':48, 'Xiaomi Model B': 1980, 'Nokia Model C':10, 'Sony Model D':0} using assert.

def test\_sell\_stock\_as\_dict(self):

self.inventory.sell\_stock({'iPhone Model A': 2, 'Xiaomi Model B': 20, 'Sony Model D': 1})

assert self.inventory.balance\_inventory == {'iPhone Model A': 48, 'Xiaomi Model B': 1980, 'Nokia Model C': 10,

'Sony Model D': 0}

# Define a pytest test method \*\*'test\_sell\_stock\_as\_list'\*\*, which tries selling the requested stock ['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'] from the existing inventory, and which checks if the method raises a TypeError with the message "Requested stock must be a dictionary" using assert.

def test\_sell\_stock\_as\_list(self):

with pytest.raises(TypeError):

MobileInventory.sell\_stock(['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'])

assert "Requested stock must be a dictionary" == balance\_inventory.sell\_stock()

# Define a pytest test method \*\*'test\_sell\_stock\_with\_numeric\_keys'\*\*, which tries selling the requested stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} from the existing inventory, and which checks if the method raises ValueError with the message "Mobile model name must be a string" using assert.

def test\_sell\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_with\_nonnumeric\_values'\*\*, which tries selling the requested stock {'iPhone Model A':'2', 'Xiaomi Model B':'3', 'Nokia Model C':'4'} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_with\_float\_values'\*\*, which tries selling the requested stock {'iPhone Model A':2.5, 'Xiaomi Model B':3.1, 'Nokia Model C':4} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_float\_values(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_of\_nonexisting\_model'\*\*, which tries selling the requested stock {'iPhone Model B':2, 'Xiaomi Model B':5} from the existing inventory, and which checks if the method raises an InsufficientException with the message "No Stock. New Model Request" using assert.

def test\_sell\_stock\_of\_nonexisting\_model(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

# Define a pytest test method \*\*'test\_sell\_stock\_of\_insufficient\_stock'\*\*, which tries selling the requested stock {'iPhone Model A':2, 'Xiaomi Model B':5, 'Nokia Model C': 15} from the existing inventory, and which checks if the method raises an InsufficientException with the message "Insufficient Stock" using assert.

def test\_sell\_stock\_of\_insufficient\_stock(self):

with pytest.raises(ValueError) as excinfo:

MobileInventory({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" in str(excinfo.value)

15 test case pass

import sys

import os

sys.path.append(os.getcwd())

from proj.inventory import MobileInventory, InsufficientException

import pytest

# Import MobileInventory class and InsufficientException from the inventory module using the expression from proj.inventory import MobileInventory, InsufficientException.

# Import pytest using the expression import pytest.

# Use assert statement for assert, and to check. Ex: assert 1 == 1

# Define a pytest test class \*\*'TestingInventoryCreation'\*\*

class TestingInventoryCreation:

# Define a pytest test method \*\*'test\_creating\_empty\_inventory'\*\*, which creates an empty inventory and checks if its 'balance\_inventory' is an empty dict using assert.

def test\_creating\_empty\_inventory(self):

c1 = MobileInventory()

assert c1.balance\_inventory == {}

# Define a pytest test method \*\*'test\_creating\_specified\_inventory'\*\*, which checks if inventory instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25}.

def test\_creating\_specified\_inventory(self):

c2 = MobileInventory({'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25})

assert c2.balance\_inventory == {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25}

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_list'\*\*, which checks if the method raises a TypeError with the message "Input inventory must be a dictionary" when a list "['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z']" is passed as input using assert.

def test\_creating\_inventory\_with\_list(self):

with pytest.raises(TypeError) :

c3 = MobileInventory(['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'])

assert "Input inventory must be a dictionary" == c3.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_numeric\_keys'\*\*, which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert, when the dict {1:'iPhone Model X', 2:'Xiaomi Model Y', 3:'Nokia Model Z'} is passed as input.

def test\_creating\_inventory\_with\_numeric\_keys(self):

with pytest.raises(ValueError):

c4 = MobileInventory({1:'iPhone Model X', 2:'Xiaomi Model Y', 3:'Nokia Model Z'})

assert "Mobile model name must be a string" == c4.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_nonnumeric\_values'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':'100', 'Xiaomi Model Y': '1000', 'Nokia Model Z':'25'} is passed as input.

def test\_creating\_inventory\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError):

c5 = MobileInventory({'iPhone Model X':'100', 'Xiaomi Model Y': '1000', 'Nokia Model Z':'25'})

assert "No. of mobiles must be a positive integer" == c5.balance\_inventory

# Define a pytest test method \*\*'test\_creating\_inventory\_with\_negative\_value'\*\*, which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert, when the dict {'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25} is passed as input.

def test\_creating\_inventory\_with\_negative\_value(self):

with pytest.raises(ValueError) :

c6 = MobileInventory({'iPhone Model X':-45, 'Xiaomi Model Y': 200, 'Nokia Model Z':25})

assert "No. of mobiles must be a positive integer" == c6.balance\_inventory

# Define another pytest test class \*\*'TestInventoryAddStock'\*\*, which tests the behavior of the \*\*'add\_stock'\*\* method, with the following tests

class TestInventoryAddStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with input {'iPhone Model X':100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25} and assign it to class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory({'iPhone Model X': 100, 'Xiaomi Model Y': 1000, 'Nokia Model Z':25})

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_dict'\*\*, which adds the new stock {'iPhone Model X':50, 'Xiaomi Model Y': 2000, 'Nokia Model A':10} to the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also, check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model X':150, 'Xiaomi Model Y': 3000, 'Nokia Model Z':25, 'Nokia Model A':10} using assert.

def test\_add\_new\_stock\_as\_dict(self):

self.inventory.add\_stock({'iPhone Model X': 50, 'Xiaomi Model Y': 2000, 'Nokia Model A': 10})

assert self.inventory.balance\_inventory == {'iPhone Model X': 150, 'Xiaomi Model Y': 3000, 'Nokia Model Z': 25, 'Nokia Model A': 10}

# Define a pytest test method \*\*'test\_add\_new\_stock\_as\_list'\*\*, which adds the new stock ['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'] to the existing inventory, and which checks if the method raises a TypeError with the message "Input stock must be a dictionary" using assert.

def test\_add\_new\_stock\_as\_list(self):

with pytest.raises(TypeError) :

MobileInventory.add\_stock(['iPhone Model X', 'Xiaomi Model Y', 'Nokia Model Z'])

assert "Input stock must be a dictionary" == balance\_inventory.add\_stock()

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_numeric\_keys'\*\*, which adds the new stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} to the existing inventory, and which checks if the method raises a ValueError with the message "Mobile model name must be a string" using assert.

def test\_add\_new\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError):

MobileInventory.add\_stock({1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'})

assert "Mobile model name must be a string" == balance\_inventory.add\_stock()

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_nonnumeric\_values'\*\*, which adds the new stock {'iPhone Model A':'50', 'Xiaomi Model B':'2000', 'Nokia Model C':'25'} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError):

MobileInventory.add\_stock({'iPhone Model A':'50', 'Xiaomi Model B': '2000', 'Nokia ModelC':'25'})

assert "Mobile model name must be a string" == balance\_inventory.add\_stock()

# Define a pytest test method \*\*'test\_add\_new\_stock\_with\_float\_values'\*\*, which adds the new stock {'iPhone Model A':50.5, 'Xiaomi Model B':2000.3, 'Nokia Model C':25} to the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_add\_new\_stock\_with\_float\_values(self):

with pytest.raises(ValueError):

MobileInventory.add\_stock({'iPhone Model A':50.5, 'Xiaomi Model B':2000.3, 'Nokia Model C':25})

assert "Mobile model name must be a string" == balance\_inventory.add\_stock()

# Define another pytest test class \*\*'TestInventorySellStock'\*\*, which tests the behavior of the \*\*'sell\_stock'\*\* method, with the following tests

class TestInventorySellStock:

inventory = None

@classmethod

# Define a pytest class fixture 'setup\_class', which creates an \*\*'MobileInventory'\*\* instance with the input {'iPhone Model A':50, 'Xiaomi Model B': 2000, 'Nokia Model C':10, 'Sony Model D':1}, and assign it to the class attribute \*\*'inventory'\*\*.

def setup\_class(cls):

cls.inventory = MobileInventory(

{'iPhone Model A': 50, 'Xiaomi Model B': 2000, 'Nokia Model C': 10, 'Sony Model D': 1})

# Define a pytest test method \*\*'test\_sell\_stock\_as\_dict'\*\*, which sells the requested stock {'iPhone Model A':2, 'Xiaomi Model B':20, 'Sony Model D':1} from the existing inventory, and update the \*\*balance\_inventory\*\* attribute. Also check if the updated \*\*balance\_inventory\*\* equals {'iPhone Model A':48, 'Xiaomi Model B': 1980, 'Nokia Model C':10, 'Sony Model D':0} using assert.

def test\_sell\_stock\_as\_dict(self):

self.inventory.sell\_stock({'iPhone Model A': 2, 'Xiaomi Model B': 20, 'Sony Model D': 1})

assert self.inventory.balance\_inventory == {'iPhone Model A': 48, 'Xiaomi Model B': 1980, 'Nokia Model C': 10,

'Sony Model D': 0}

# Define a pytest test method \*\*'test\_sell\_stock\_as\_list'\*\*, which tries selling the requested stock ['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'] from the existing inventory, and which checks if the method raises a TypeError with the message "Requested stock must be a dictionary" using assert.

def test\_sell\_stock\_as\_list(self):

with pytest.raises(TypeError):

MobileInventory.sell\_stock(['iPhone Model A', 'Xiaomi Model B', 'Nokia Model C'])

assert "Requested stock must be a dictionary" == balance\_inventory.sell\_stock()

# Define a pytest test method \*\*'test\_sell\_stock\_with\_numeric\_keys'\*\*, which tries selling the requested stock {1:'iPhone Model A', 2:'Xiaomi Model B', 3:'Nokia Model C'} from the existing inventory, and which checks if the method raises ValueError with the message "Mobile model name must be a string" using assert.

def test\_sell\_stock\_with\_numeric\_keys(self):

with pytest.raises(ValueError):

MobileInventory.sell\_stock({1: 'iPhone Model A', 2: 'Xiaomi Model B', 3: 'Nokia Model C'})

assert "Mobile model name must be a string" == balance\_inventory.sell\_stock()

# Define a pytest test method \*\*'test\_sell\_stock\_with\_nonnumeric\_values'\*\*, which tries selling the requested stock {'iPhone Model A':'2', 'Xiaomi Model B':'3', 'Nokia Model C':'4'} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_nonnumeric\_values(self):

with pytest.raises(ValueError):

MobileInventory.sell\_stock({'iPhone Model A': '5', 'Xiaomi Model B': '3', 'Nokia Model C': '4'})

assert "No. of mobiles must be a positive integer" == balance\_inventory.sell\_stock()

# Define a pytest test method \*\*'test\_sell\_stock\_with\_float\_values'\*\*, which tries selling the requested stock {'iPhone Model A':2.5, 'Xiaomi Model B':3.1, 'Nokia Model C':4} from the existing inventory, and which checks if the method raises a ValueError with the message "No. of mobiles must be a positive integer" using assert.

def test\_sell\_stock\_with\_float\_values(self):

with pytest.raises(ValueError):

MobileInventory.sell\_stock({'iPhone Model A': 2.5, 'Xiaomi Model B': 3.1, 'Nokia Model C': 4})

assert "No. of mobiles must be a positive integer" == balance\_inventory.sell\_stock()

# Define a pytest test method \*\*'test\_sell\_stock\_of\_nonexisting\_model'\*\*, which tries selling the requested stock {'iPhone Model B':2, 'Xiaomi Model B':5} from the existing inventory, and which checks if the method raises an InsufficientException with the message "No Stock. New Model Request" using assert.

def test\_sell\_stock\_of\_nonexisting\_model(self):

with pytest.raises(InsufficientException):

MobileInventory.sell\_stock({'iPhone Model B': 2, 'Xiaomi Model B': 5})

assert "No Stock. New Model Request" == balance\_inventory.sell\_stock()

# Define a pytest test method \*\*'test\_sell\_stock\_of\_insufficient\_stock'\*\*, which tries selling the requested stock {'iPhone Model A':2, 'Xiaomi Model B':5, 'Nokia Model C': 15} from the existing inventory, and which checks if the method raises an InsufficientException with the message "Insufficient Stock" using assert.

def test\_sell\_stock\_of\_insufficient\_stock(self):

with pytest.raises(InsufficientException):

MobileInventory.sell\_stock({'iPhone Model A': 2, 'Xiaomi Model B': 5, 'Nokia Model C': 15})

assert "Insufficient Stock" == balance\_inventory.sell\_stock()