# task\_manager.py

class Task:

def \_\_init\_\_(self, task\_id, description):

self.task\_id = task\_id

self.description = description

self.is\_completed = False

def complete(self):

self.is\_completed = True

def \_\_repr\_\_(self):

status = "Done" if self.is\_completed else "Pending"

return f"Task({self.task\_id}, {self.description}, {status})"

class TaskManager:

def \_\_init\_\_(self):

self.tasks = {} # Dictionary to store tasks with task\_id as key.

self.next\_id = 1

def add\_task(self, description):

task = Task(self.next\_id, description)

self.tasks[self.next\_id] = task

self.next\_id += 1

return task

def remove\_task(self, task\_id):

if task\_id in self.tasks:

del self.tasks[task\_id]

else:

raise ValueError("Task not found")

def complete\_task(self, task\_id):

if task\_id in self.tasks:

self.tasks[task\_id].complete()

else:

raise ValueError("Task not found")

def get\_all\_tasks(self):

return list(self.tasks.values())

Test Cases:

# test\_task\_manager.py

import unittest

import timeit

from task\_manager import Task, TaskManager

# -------------------------------

# Test A

# -------------------------------

class TestA(unittest.TestCase):

def setUp(self):

self.manager = TaskManager()

def test\_add\_and\_complete\_task\_integration(self):

task = self.manager.add\_task("Write integration tests")

self.manager.complete\_task(task.task\_id)

self.assertTrue(task.is\_completed)

def test\_remove\_task\_integration(self):

task = self.manager.add\_task("Task to remove")

self.manager.remove\_task(task.task\_id)

self.assertNotIn(task.task\_id, self.manager.tasks)

# -------------------------------

# Test B

# -------------------------------

class TestB(unittest.TestCase):

def setUp(self):

self.manager = TaskManager()

def test\_full\_task\_flow(self):

task1 = self.manager.add\_task("Buy groceries")

task2 = self.manager.add\_task("Call John")

task3 = self.manager.add\_task("Pay bills")

self.manager.complete\_task(task2.task\_id)

tasks = self.manager.get\_all\_tasks()

self.assertEqual(len(tasks), 3)

completed\_tasks = [t for t in tasks if t.is\_completed]

self.assertEqual(len(completed\_tasks), 1)

self.assertEqual(completed\_tasks[0].task\_id, task2.task\_id)

# -------------------------------

# Test C

# -------------------------------

class TestC(unittest.TestCase):

def setUp(self):

self.manager = TaskManager()

def test\_add\_tasks\_performance(self):

# Measure performance of adding 10,000 tasks.

def add\_tasks():

for i in range(10000):

self.manager.add\_task(f"Task {i}")

elapsed = timeit.timeit(add\_tasks, number=1)

# Arbitrary performance threshold (e.g., under 1 second)

self.assertLess(elapsed, 1.0, f"Adding tasks took too long: {elapsed} seconds")

# -------------------------------

# Test D

# -------------------------------

class TestD(unittest.TestCase):

def setUp(self):

self.manager = TaskManager()

def test\_remove\_nonexistent\_task\_regression(self):

with self.assertRaises(ValueError):

self.manager.remove\_task(999)

def test\_complete\_nonexistent\_task\_regression(self):

with self.assertRaises(ValueError):

self.manager.complete\_task(888)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()