**Assignment 12**

from \_\_future\_\_ import annotations

from abc import ABC, abstractmethod

class Client:

# content = None

def \_\_init\_\_(self, inp, init\_state):

if init\_state == "1":

context = Context(ConcreteState1())

elif init\_state == "2":

context = Context(ConcreteState2())

elif init\_state == "3":

context = Context(ConcreteState3())

else:

context = Context(ConcreteState4())

self.inp = inp

self.inp = self.inp.split(" ")

for i in self.inp:

if i == "a":

context.request1()

elif i == "b":

context.request2()

class Context:

\_state = None

def \_\_init\_\_(self, state: State) -> None:

self.transition\_to(state)

def transition\_to(self, state: State):

print(f"Transition to {type(state).\_\_name\_\_}")

self.\_state = state

# print("Current state: ", state)

self.\_state.context = self

# print("State context = ", self)

def request1(self):

self.\_state.OnA()

def request2(self):

self.\_state.OnB()

class State(ABC):

@property

def context(self) -> Context:

return self.\_context

@context.setter

def context(self, context: Context) -> None:

self.\_context = context

@abstractmethod

def OnA(self) -> None:

pass

@abstractmethod

def OnB(self) -> None:

pass

class ConcreteState1(State):

def OnA(self) -> None:

# print("ConcreteState1 handles OnA.")

# print("ConcreteState1 wants to change the state of the context.")

self.context.transition\_to(ConcreteState2())

def OnB(self) -> None:

# print("ConcreteState1 handles OnB.")

# print("ConcreteState1 wants to change the state of the context.")

self.context.transition\_to(ConcreteState3())

class ConcreteState2(State):

def OnA(self) -> None:

# print("ConcreteState2 handles OnA.")

# print("ConcreteState2 wants to change the state of the context.")

self.context.transition\_to(ConcreteState1())

def OnB(self) -> None:

# print("ConcreteState2 handles OnB.")

# print("ConcreteState2 wants to change the state of the context.")

self.context.transition\_to(ConcreteState4())

class ConcreteState3(State):

def OnA(self) -> None:

# print("ConcreteState3 handles OnA.")

# print("ConcreteState3 wants to change the state of the context.")

self.context.transition\_to(ConcreteState4())

def OnB(self) -> None:

# print("ConcreteState3 handles OnB.")

# print("ConcreteState3 wants to change the state of the context.")

self.context.transition\_to(ConcreteState1())

class ConcreteState4(State):

def OnA(self) -> None:

# print("ConcreteState4 handles OnA.")

# print("ConcreteState4 wants to change the state of the context.")

self.context.transition\_to(ConcreteState3())

def OnB(self) -> None:

# print("ConcreteState4 handles OnB.")

# print("ConcreteState4 wants to change the state of the context.")

self.context.transition\_to(ConcreteState2())

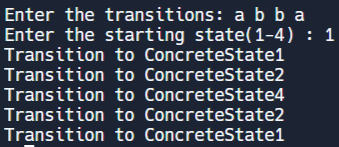
if \_\_name\_\_ == "\_\_main\_\_":

inp = input("Enter the transitions: ")

init\_state = input("Enter the starting state(1-4) : ")

Client(inp, init\_state)

**Output:**

****