MEC INFOTECH VIETNAM

MATSUSAKA EDP CENTER INFOTECH VIETNAM CO.,LTD

RECRUITMENT EXAM

February 23, 2024

Python Programming

[Direction]
Read the description of the following program then answer the question.
The time limit is 25 minutes.

<Description of the program>
This is an electronic meeting system. This program allows people to communicate with each other when they have connected to the server. If you want to join a meeting you connect to the server and log in. The system generates a Client for you. Messages from logged-in people are delivered to the server. After receiving the messages the server distributes them to all the Clients. When you want to finish a meeting you log out. The following classes need to be implemented.

(1) The MessageQueue class serves as a message queue, maintaining the order of messages based on a First In First Out (FIFO) principle. The queue has a maximum size (MAX_SIZE), and the put method is responsible for adding a message to the queue. If the queue is full, the method gracefully waits until space becomes available. The take method retrieves a message from the head of the queue, waiting if the queue is currently empty.

(2) Session class: The Session class represents a client's session, allowing clients to write messages and log out. It is initialized with a reference to the server.

(3) The ConfServer class manages the server, including a message queue, session table, and server thread. It follows the Singleton pattern, ensuring only one instance exists. The server runs in a continuous loop, taking messages from the queue and distributing them to all connected clients. Clients log in using the login method, creating a session and registering it in the session table. The write_message method sends a message to the server, which is then added to the message queue. The deliver_message method broadcasts messages to all clients. The logout_impl method removes a client session from the session table.

(3) The ConfClient class serves as an abstract base class for clients in the electronic meeting system, providing essential functions required by the server. Upon initialization with a non-null name, clients gain access to methods such as display_message for customized message handling. The distinctive feature is the write_message method, enabling clients to send messages to the server via a ConfServer.Session instance obtained during login. Additionally, the logout method facilitates the logout process. The class incorporates equality and hashing methods based on client names. It serves as a blueprint, demanding concrete implementations of display_message, write_message, and logout in subclasses to define specific client behavior within the collaborative communication system.

(4) When you test the server functions you need to implement the *TestClient* class. The TestClient class is a specific implementation of a client for testing purposes. It extends ConfClient and implements the display_message method to print messages in a specific format. An example is provided in the usage section, demonstrating client login, message exchange, and logout. The *displayMessage* method outputs a message as the following format.

Source Client Name: Message -> Destination Client Name

```
from threading import Lock, Condition
from threading import Thread
from collections import defaultdict
import time

class Message:
    def __init__(self, sender, content):
        self.sender = sender
        self.content = content

    def __str__(self):
        return f"{self.sender}: {self.content}"

class MessageQueue:
    MAX_SIZE = 3

    def __init__(self):
        self.queue = []
        self.lock = Lock()
        self.not_empty = Condition(self.lock)
        self.not_full = Condition(self.lock)

    def put(self, message: Message) -> None:
        with self.lock:
            while a
                self.not_full.wait()
            self.queue.append(message)
            self.not_empty.notify_all()

    def take(self) -> Message:
        with self.lock:
            while len(self.queue) == 0:
                self.not_empty.wait()
            message = self.queue.pop(0)
            self.not_full.notify_all()
            return message
```

```
class Session:
    def __init__(self, server):
        self.server = server

    def write_message(self, message_content):
        self.server.write_message(self, message_content)

    def logout(self):
        self.server.logout_impl(self)

class ConfClient:
    def __init__(self, name):
        if name is None:
            raise ValueError("Name cannot be None")
        self.name = name

    def get_name(self):
        return self.name

    def display_message(self, message):
        pass  # Implementation specific to each client subclass

    def __eq__(self, other):
        if not isinstance(other, ConfClient):
            return False
        return self.name == other.name

    def __hash__(self):
        return hash(self.name)
```

```
class ConfServer:
    _server = None

    def __init__(self):
        self.queue = MessageQueue()
        self.sessions_table = defaultdict()
        self.instance = self

        # Start the server thread
        self.server_thread = Thread(target=[ b ])
        self.server_thread.start()

    def logout_impl(self, session):
        if session in self.sessions_table:
            del self.sessions_table[session]

    @classmethod
    def login(cls, client):
        if client is None:
            raise ValueError("Client cannot be None")
        return cls.get_server().login_impl(client)

    @classmethod
    def get_server(cls):
        if cls._server is None:
            cls._server = cls()
        return cls._server

    def run(self):
        while True:
            message = self.queue.[ c ]
            self.deliver_message(message)

      def login_impl(self, client):
          session = Session(self)
          self.sessions_table[session] = client
          return session
```

```
    def write_message(self, session, message_content):
        client = self.get_client(session)
        message = Message(client.get_name(), [ d ] )
        self.queue.put(message)

    def deliver_message(self, message):
        for session in self.sessions_table.keys():
            self.sessions_table[session].display_message(str(message))

    def get_client(self, session):
        if [ e ] :
            raise ValueError("Invalid Session")
        return self.sessions_table[session]

class TestClient(ConfClient):
    def __init__(self, name):
        super().__init__(name)

    def display_message(self, message):
        print(message + " >" + self.get_name())

# Usage example:
if __name__ == "__main__":
    bob = ConfServer.login(TestClient("Bob"))
    sam = ConfServer.login(TestClient("Sam"))

    bob.write_message("Hello. This is Bob.")
    sam.write_message("Hello.")
    bob.write_message("How are you?")
    sam.write_message("I'm good. Thank you.")

    time.sleep(1)

    bob.logout()
    sam.logout()
```

1. Select the best answer for [a]

   A: len(self.queue) > 0
   B: len(self.queue) == 0
   C: len(self.queue) >= self.MAX_SIZE
   D: len(self.queue) < self.MAX_SIZE
   E: len(self.queue) <= self.MAX_SIZE
   F: len(self.queue) != 0

2. Select the best answer for [b]

   A: self.run_server
   B: self.run
   C: ConfServer.run
   D: self.run_server
   E: self.start_server
   F: self.server.run

3. Select the best answer for [c]

   A: get()
   B: take()
   C: pop()
   D: removeFirst()
   E: dequeue()
   F: remove()

4. Select the best answer for [d]

   A: client.get_message_content()
   B: message
   C: client.get_message()
   D: message_content

5. Select the best answer for | e |

A: session in self.sessions_table
B: session not in self.sessions_table
C: len(self.sessions_table) == 0
D: self.sessions_table[session] is not None
E: session is not None
F: len(self.sessions_table) > 0