問 11　次の Java プログラムの説明及びプログラムを読んで，設問 1，2 に答えよ。

（Java プログラムで使用する API の説明は，この冊子の末尾を参照してください。）

〔プログラムの説明〕

電子会議システムのプログラムである。

電子会議は，サーバに接続しているクライアント間で行われ，一つのクライアントは，会議の参加者 1 人に対応する。会議に参加するときは，電子会議システムのサーバにログインする。参加者の発言はクライアントからのメッセージとしてサーバに送られる。サーバは受信したメッセージを，発信元クライアントを含む全クライアントに配信する。会議から退出するときは，ログアウトする。

この電子会議システムのサーバを実装するために，次のクラスを定義する。

(1) クラス MessageQueue は，クライアントからのメッセージを格納するための待ち行列（メッセージキュー）である。メッセージは，先入れ先出しで管理される。メソッド put は，引数 message で与えられたメッセージをメッセージキューに追加する。メッセージキューに格納できるメッセージ数には制限があり，満杯のときは，空きができるまで待つ。メソッド take は，メッセージキューの先頭からメッセージを取り出す。メッセージキューが空のときは，メッセージが追加されるまで待つ。

(2) クラス ConfServer は，サーバを定義する。クライアントとのログイン状態を管理し，メッセージの受信と配信を行う。ここでクライアントとは，(3) で説明する抽象クラスの型のインスタンスである。クラス ConfServer は，クラスが初期化されるときにインスタンスが作成され，単独のスレッドとして動作する。メソッド run は，メッセージキューからメッセージを取り出し，ログインしている全クライアントにそのメッセージを配信する。この操作を繰り返す。メソッド login は，入れ子クラス ConfServer.Session のインスタンスを生成し，それをキーとしてクライアントを管理テーブルに登録し，そのインスタンスを返す。クライアントが既に登録されている場合は，IllegalArgumentException を投げる。クライアントが管理テーブルに登録されているとき，そのクライアントはサーバにログインしている状態であるとする。

(3) 抽象クラス ConfClient は，サーバが必要とするクライアントの機能を定義する。サーバは，メソッド displayMessage を呼び出してクライアントにメッセージを配

信する。クライアントは，このクラスを実装しなければならない。クライアントがサーバにメッセージを送信するときは，ログイン時に返されたクラス ConfServer.Session のインスタンスのメソッド writeMessage を呼び出す。クライアントがログアウトするときは，同じインスタンスのメソッド logout を呼び出す。

(4) サーバをテストするために，クラス TestClient を定義する。このクラスは，ConfClient で定義されたメソッドをテスト用に実装する。TestClient のインスタンスは，サーバに対してクライアントの役割をする。メソッド displayMessage は，メッセージを次の形式で出力する。

発信クライアント名: メッセージ　>受信クライアント名

〔プログラム 1〕

```
import java.util.LinkedList;

public class MessageQueue {
   // 格納できる最大のメッセージ数
   private final static int MAX_SIZE = 3;
   private final LinkedList<String> queue = new LinkedList<String>();

   public synchronized void put(String message) {
      while (queue.size() >= MAX_SIZE) {
         try {
            wait();
         } catch (InterruptedException e) { }
      }
      queue.add(message);
      notifyAll();
   }

   public synchronized String take() {
      while ( [ a ] ) {
         try {
            wait();
         } catch (InterruptedException e) { }
      }
      String message = queue.removeFirst();
      notifyAll();
      return message;
   }
}
```

〔プログラム2〕

```
import java.util.HashMap;
import java.util.Map;

public class ConfServer implements Runnable {
   // ConfServer のインスタンス
   private static final ConfServer server;
   static {
      server = new ConfServer();
      new Thread(server).start();
   }

   // クライアントからのメッセージを格納するメッセージキュー
   private final MessageQueue queue = new MessageQueue();

   // セッション管理テーブル
   private final Map<Session, ConfClient> sessionsTable
         = new HashMap<Session, ConfClient>();

   public static Session login(ConfClient client) {
      if (client == null)
         throw new NullPointerException();
      return server.loginImpl(client);
   }

   private ConfServer() { }

   public void run() {
      while (true) {
         String message = queue.take();
         deliverMessage(message);
      }
   }

   private void writeMessage(Session session,
                             String message) {
      ConfClient client = getClient(session);
      String s = client.getName() + ": " + message;
      queue.put(s);
   }

   private synchronized void deliverMessage(String message) {
      for (Session session : sessionsTable.keySet())
         [ b ].displayMessage(message);
   }

   private synchronized ConfClient getClient(Session session) {
      ConfClient client = sessionsTable.get(session);
      if (client == null)
         throw new IllegalStateException("無効なセッション");
      return client;
   }
```

```
    private synchronized Session loginImpl(ConfClient client) {
        if (sessionsTable.containsValue(client))
            throw new IllegalArgumentException(
                    client.getName() + "はログイン済み");
        Session session = new Session();
        sessionsTable.put(session, client);
        return session;
    }

    private synchronized void logoutImpl(Session session) {
        sessionsTable.remove(session);
    }

    public static class Session {
        private Session() { }

        public void writeMessage(String msg) {
            server.writeMessage(this, msg);
        }

        public void logout() {
            server.logoutImpl( c );
        }
    }
}
```

〔プログラム3〕

```
public abstract class ConfClient {
    private final String name;

    public ConfClient(String name) {
        if (name == null)
            throw new NullPointerException();
        this.name = name;
    }

    public final String getName() { return name; }

    public abstract void displayMessage(String message);

    public boolean equals(Object obj) {
        if (!(obj instanceof ConfClient))
            return false;
        return name.equals(((ConfClient)obj).name);
    }

    public int hashCode() {
        return name.hashCode();
    }
}
```

〔プログラム4〕

```
public class TestClient [ d ] ConfClient {
   TestClient(String name) { [ e ]; }

   public void displayMessage(String message) {
      System.out.println(message + " >" + getName());
   }

   public static void main(String[] arg) {
      ConfServer.Session yamada
              = ConfServer.login(new TestClient("山田"));
      ConfServer.Session sato
              = ConfServer.login(new TestClient("佐藤"));

      yamada.writeMessage("こんにちは。山田です。");
      sato.writeMessage("こんにちは。");
      yamada.writeMessage("開発プランの資料は届いていますか。");
      sato.writeMessage("はい、手元にあります。");
      yamada.writeMessage("では、資料に沿ってご説明します。");

      // 全メッセージを配信し終わるように，1秒間待つ。
      try {
         Thread.sleep(1000);
      } catch (InterruptedException e) { }
      yamada.logout();
      sato.logout();
   }
}
```

Java

設問1　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

aに関する解答群

ア　!queue.isEmpty()　　イ　queue.isEmpty()

ウ　queue.size() != MAX_SIZE　　エ　queue.size() < MAX_SIZE

オ　queue.size() == MAX_SIZE

bに関する解答群

ア　session.get(message)　　イ　session.get(server)

ウ　session.get(sessionsTable)　　エ　sessionsTable.get(message)

オ　sessionsTable.get(server)　　カ　sessionsTable.get(session)

cに関する解答群

ア　queue　　イ　server　　ウ　session

エ　sessionsTable　　オ　this

dに関する解答群

ア　abstract　　イ　extends　　ウ　implements

エ　static　　オ　throws

eに関する解答群

ア　super()　　イ　super(name)　　ウ　super(this)

エ　this()　　オ　this(name)

設問2　クラスMessageQueueのメソッドput及びtakeには，synchronized修飾子が付けられている。これには理由が二つある。一つは，キューの状態（空又は満杯）によってスレッド間でメソッドwait及びnotifyAllを呼び出して同期を取るためである。もう一つの理由として適切な答えを，解答群の中から選べ。

解答群

ア　クラスConfClientのインスタンスがそれぞれ別スレッドとして同時に動くことを想定すると，一度にフィールドqueueで参照されるLinkedListのインスタンスにアクセスが集中することが考えられる。そこで，一時点でqueueにアクセスできるスレッドを一つにして，システムに負荷がかかりすぎるのを防ぐ。

イ　クラスConfClientのインスタンスがそれぞれ別スレッドとして同時に動くことを想定すると，タイミングによってはフィールドqueueで参照されるLinkedListのインスタンスに格納されている他のスレッドが書き込んだメッセージを読み出すことができ，セキュリティ上問題となる。そこで，一時点でqueueにアクセスできるスレッドを一つにして，queueに既に書き込まれているメッセージを読み出せないようにする。

ウ　クラスConfClientのインスタンスとクラスConfServerのインスタンスは別スレッドとして動くので，フィールドqueueで参照されるLinkedListのインスタンスに対しての操作が同時に実行されることがある。そこで，一時点でqueueにアクセスできるスレッドを一つにして，データの内部状態に矛盾が起きないようにする。

エ　クラスConfClientのインスタンスの個数が多いとフィールドqueueで参照されるLinkedListのインスタンスに格納するメッセージ数の増大によってメモリ不足によるエラーが発生し，ConfServerを実行しているスレッドが停止する可能性がある。そこで，ConfClientのインスタンスの個数を制限して，エラーを起こさないようにする。