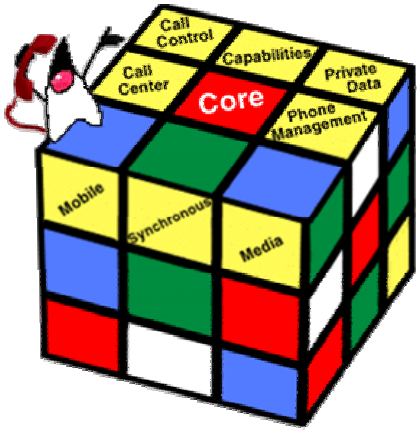
PHẦN IV.
LẬP TRÌNH TRUYỀN THÔNG QUA MẠNG PSTN&ET
CHƯƠNG 5
LẬP TRÌNH ỨNG DỤNG TRUYỀN THÔNG QUA MẠNG ĐIỆN THOẠI CÔNG CỘNG (PSTN)

I. KỸ THUẬT LẬP TRÌNH VỚI JTAPI

1. Giới thiệu thư viện JTAPI

JTAPI là một giao diện lập trình ứng dụng hướng đối tượng cho những ứng dụng máy tính-điện thoại trên nền Java. Tương tự như những giao diện lập trình ứng dụng cho các nền tảng khác như TAPI (Telephony API) trên Microsoft Windows và TSAPI trên Novell Netware. Cấu trúc của thư viện JTAPI được thể hiện như hình sau: Nó gồm bộ cốt lõi và các gói mở rộng chuẩn.



Hình 6.1. Cấu trúc thư viện JTAPI

Tại trung tâm của JTAPI là gói "cốt lõi ". Gói cốt lõi cung cấp khung cơ bản cho mô hình gọi điện thoại và những đặc trưng điện thoại sơ khai ban đầu . Những đặc tính này bao gồm định vị một cuộc gọi, trả lời một gọi, và huỷ một cuộc gọi. Những ứng dụng kỹ thuật điện thoại đơn giản sẽ chỉ cần sử dụng lõi để thực hiện các tác vụ của chúng mà không cần quan tâm tới những chi tiết của những gói khác. Chẳng hạn, gói lõi cho phép người sử dụng dễ dàng thiết kế để thêm đặc tính điện thoại vào một trang Web.

Phân tầng xung quanh gói lõi JTAPI là một số gói "mở rộng chuẩn ". Những gói mở rộng này bổ sung thêm các chức năng điện thoại cho API. Các gói mở rộng chuẩn trong API bao gồm các gói sau:*callcontrol, callcenter, media, phone, privatepackages* và gói *capabilities.*

- *Gói điều khiển gọi – call control.*

Gói *javax.telephony.callcontrol*: Mở rộng lõi bằng việc cung cấp các cuộc gọi mức cao hơn bao gồm các đặc tính điều khiển điện thoại như giữ cuộc gọi, chuyển cuộc gọi... Gói này cũng cung cấp một mô hình trạng thái chi tiết hơn của những cuộc gọi. Các lớp tiêu biểu của gói gồm các giao diện sau:

- CallControlAddress
- CallControlAddressObserver
- CallControlCall
- CallControlCallObserver

- CallControlConnection
- CallControlTerminal
- CallControlTerminalConnection
- CallControlTerminalObserver

➢ *Gói callcenter*

Gói *javax.telephony.callcenter* cung cấp khả năng thực hiện quản lý các trung tâm cuộc gọi lớn ở mức độ cao. Ví dụ như: định tuyến, phân bố cuộc gọi tự động ACD, dự báo cuộc gọi và liên kết dữ liệu ứng dụng với đối tượng điện thoại. Gói này gồm các lớp sau:

- ACDAddress
- ACDAddressObserver
- ACDConnection
- ACDManagerAddress
- ACDManagerConnection
- AgentTerminal
- AgentTerminalObserver
- CallCenterAddress
- CallCenterCall
- CallCenterCallObserver
- CallCenterProvider
- RouteAddress
- RouteCallback
- RouteSession

➢ *Gói Media.*

Gói *javax.telephony.media* cho phép truy nhập tới các luồng(stream) phương tiện truyền thông liên quan đến cuộc gọi. Chúng cho phép đọc và viết dữ liệu từ những luồng phương tiện truyền thông này. Gói này gồm các lớp:

- MediaCallObserver
- MediaTerminalConnection

➢ *Gói Phone:*

Gói *javax.telephony.phone* cho phép các ứng dụng điều khiển các đặc tính vật lý của phần cứng điện thoại.

Gói Phone gồm các lớp:

- Component
- ComponentGroup
- PhoneButton
- PhoneDisplay
- PhoneGraphicDisplay
- PhoneHookswitch
- PhoneLamp
- PhoneMicrophone
- PhoneRinger
- PhoneSpeaker
- PhoneTerminal
- PhoneTerminalObserver

- *Gói capabilities :*

Gói *javax. telephony.capabilities* là gói cung cấp cho các ứng dụng khả năng truy vấn tới hoạt động xác định một khi nó được thực hiện. Và nó gồm các lớp sau :

- AddressCapabilities
- CallCapabilities
- ConnectionCapabilities
- ProviderCapabilities
- TerminalCapabilities
- TerminalConnectionCapabilities

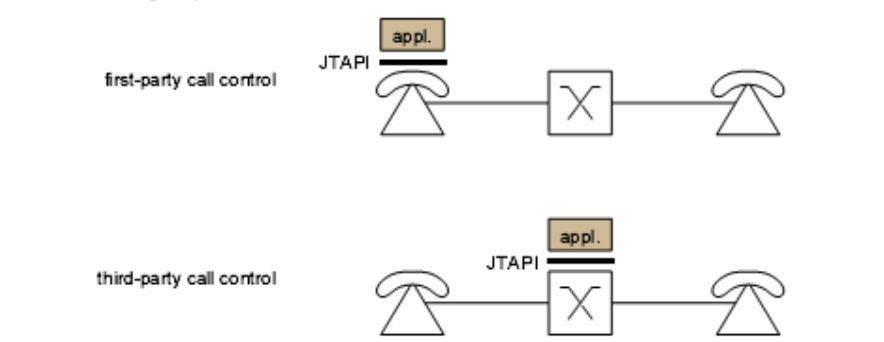
- *Gói Private Data*

Gói *javax.telephony.privatedata* cho phép các ứng dụng truyền trực tiếp dữ liệu trên các chuyển mạch cứng. Dữ liệu này được sử dụng để chỉ dẫn chuyển mạch thực hiện một thao tác chuyển mạch cụ.

2. Cơ sở của JTAPI.

Mục đích của thư viện JTAPI được xây dựng để tạo ra một giao diện cho phép trình ứng dụng Java giao tiếp với hệ thống điện thoại. Điểm giao tiếp này xác định mức độ điều khiển mà một ứng dụng phải có. JTAPI hỗ trợ cả 2 kiểu ứng dụng: first-party và third-party.

Trong ứng dụng first- party, giao diện được định vị tại thiết bị đầu cuối. Ứng dụng có cùng mức độ điều khiển như cuộc gọi điện thoại bình thường của người dùng. Trong kịch bản điều khiển third –party, giao diện được xác định bên trong hệ thống điện thoại và phụ thuộc vào hệ thống điện thoại. Sự truy cập bên trong này thường cung cấp cho ứng dụng nhiều khả năng điều khiển hơn kịch bản first- party.



Hình 6.2. Điều khiển cuộc gọi

JTAPI trong thực tế, thực chất là một tập API. Bộ cốt lõi của API cung cấp mô hình cuộc gọi cơ bản và những đặc trưng điện thoại cơ sở nhất như: định vị cuộc gọi và trả lời các cuộc gọi telephone.

Các đặc trưng của điện thoại Java là:

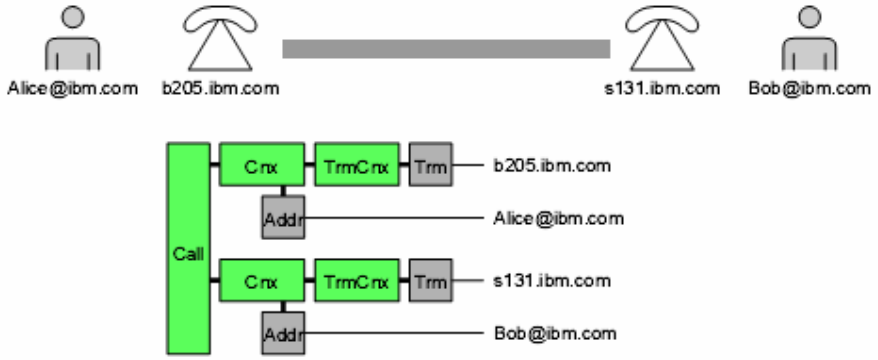
- Làm đơn giản hầu hết các ứng dụng điện thoại cơ bản
- Cung cấp một khung làm việc mà trải khắp các ứng dụng desktop đối với các ứng dụng điện thoại trung tâm gọi phân tán.
- Giao tiếp các ứng dụng trực tiếp với các nơi cung cấp dịch vụ hoặc thực hiện giao tiếp với các API điện thoại tồn tại sẵn như SunXTL, TSAPI, and TAPI.
- Dựa trên bộ lõi đơn giản, gia tăng thêm các gói mở rộng chuẩn.
- Chạy được trên một phạm vi rộng các cấu hình phần cứng một khi Java run-time được sử dụng.

3. Các cấu hình cuộc gọi tiêu biểu

Mục này trình bày những ví dụ cấu hình cuộc gọi được lựa chọn để giải thích mô hình gọi. Nó được bắt đầu với một cuộc gọi 2 phía cơ bản, sau đó mở rộng ví dụ với cuộc gọi, người sử dụng và các thiết bị đầu cuối khác.

Cuộc gọi 2 phía(two- party call):

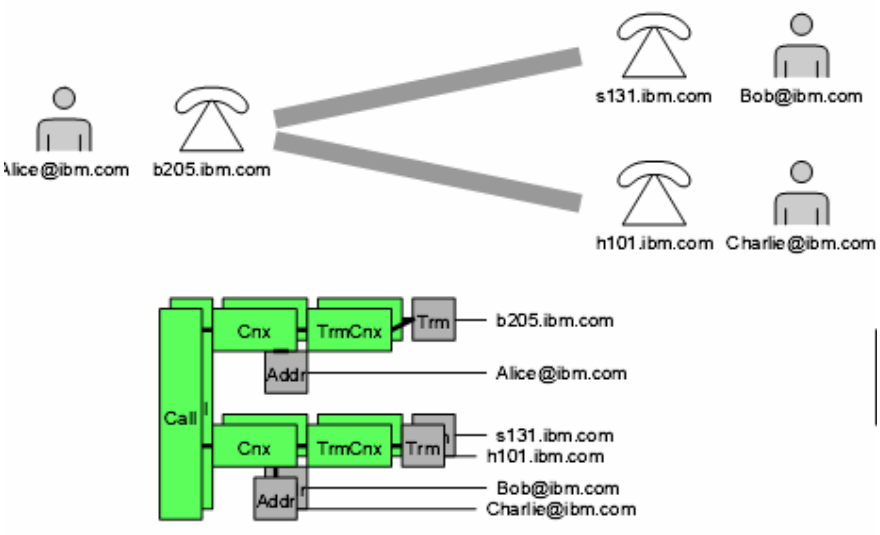
Một ví dụ cuộc gọi với hai người tham gia được biểu diễn trong hình 6.3. Những người chưa có kinh nghiệm có thể rất ngạc nhiên trong trường hợp đơn giản này: hai đối tượng kết nối (*Connection object*) gắn vào đối tượng cuộc gọi (*Call object*), mỗi đối tượng kết nối cho mỗi người tham gia. Cấu hình này cho phép mở rộng để thực hiện cho cuộc gọi hội thảo với ba hoặc nhiều người tham gia hơn. Cần chú ý rằng mô hình này hoàn toàn cân đối (Nó không phân biệt giữa thực thể cục bộ và thực thể ở xa) bởi vì nó cung cấp cách nhìn third-party



Hình 6.3.. Mô hình two- party call

Hai cuộc gọi đồng thời:

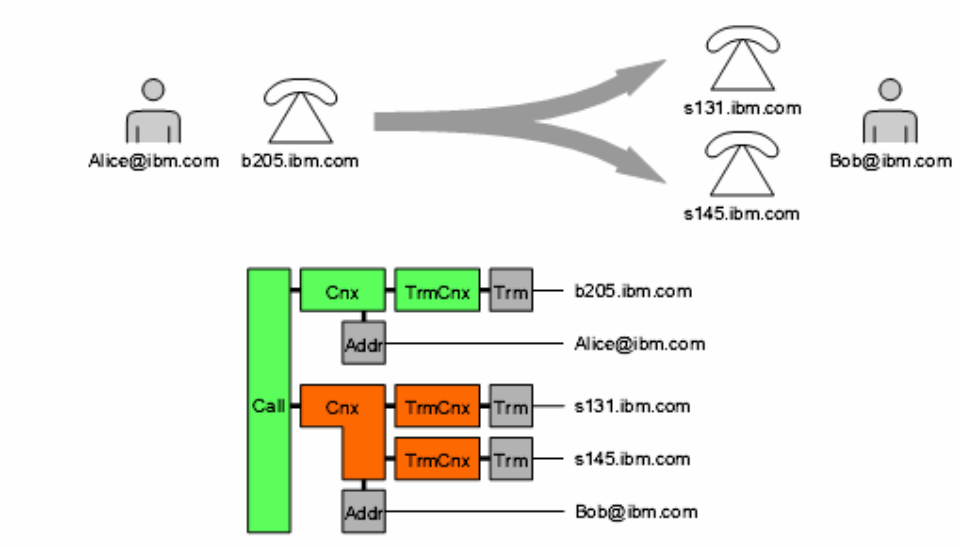
Một ví dụ về một người sử dụng mà có hai cuộc gọi đồng loạt trên cùng thiết bị đầu cuối được đưa vào hình 6.4. Mọi đối tượng liên quan cuộc gọi đã gấp đôi số của họ. Đối tượng địa chỉ (*Address object*) và đối tượng thiết bị đầu cuối (*Terminal object*) của người sử dụng có hai cuộc gọi chỉ sinh ra một lần nhưng được gán cho hai đối tượng kết nối (*Connection object*) và hai đối tượng kết nối đầu cuối (*TerminalConnection*).



Hình6.4. Mô hình Two simultaneous calls.

Cài đặt cuộc gọi với hai thiết bị đầu cuối:

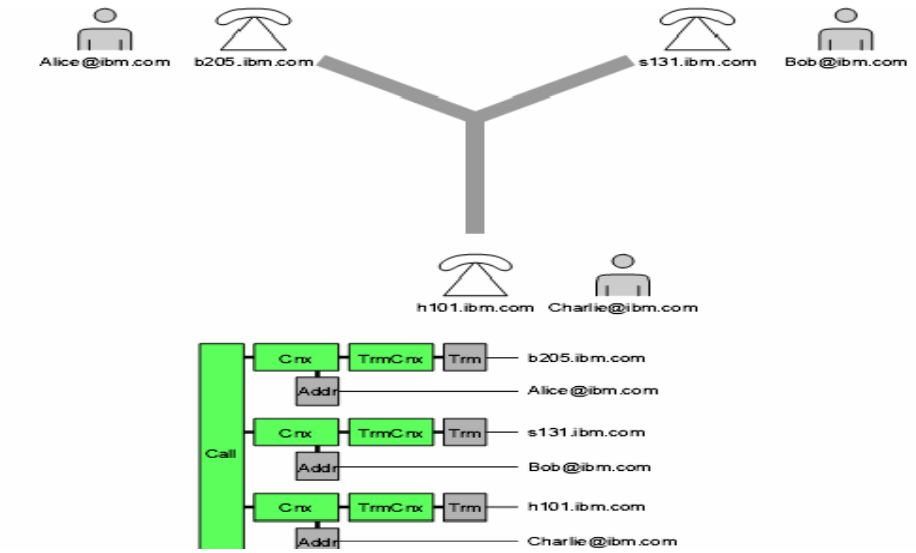
Một ví dụ cuộc gọi hai người với thiết bị đầu cuối có chuông báo được trình bày trong hình 6.5. Trong ví dụ trên, Bob thực hiện nhiều luồng, có nghĩa rằng khi Bob được gọi thì vài thiết bị đầu cuối sẽ đổ chuôngngự thể hiện nhiều luồng được đại diện bởi hai đối tượng kết nối đầu cuối gắn cho kết nối đối tượng của Bob, mỗi đối tượng cho mỗi thiết bị đầu cuối. Khi một trong những thiết bị đầu cuối trả lời cuộc gọi thì thiết bị đầu cuối khác sẽ bị loại ra(trong giới hạn của mô hình cuộc gọi này, đối tượng kết nối đầu cuối được đặt vào trong một trạng thái cấm hoạt động)



Hình 6.5. Mô hình Two alerting terminal calls.

Cuộc gọi 3 thành viên:

Một ví dụ tiêu biểu cho cuộc gọi ba thành viên là cuộc gọi hội nghị với ba người tham gia được thể hiện như hình 6.6. Mô hình cuộc gọi là một sự mở rộng trực tiếp từ mô hình cuộc gọi cơ bản với hai người tham gia. Mô hình đơn giản thêm một thành viên thứ ba với các đối tượng kết nối, địa chỉ, kết nối đầu cuối và thiết bị đầu cuối cho người thứ ba tham gia.



Hình 6.6. Mô hình Third-party call.

4. Mô hình cuộc gọi Java

4*.1. Nguyên tắc*

JTAPI là một mô hình trừu tượng hóa mức độ cao và độc lập về công nghệ. Nó mô tả cuộc gọi như là một tập hữu hạn trạng thái máy mà phải trải qua trạng thái chuyển tiếp đó khi cuộc gọi được thực hiện.

Mô hình cuộc gọi được xây dựng tổng quát, bao trùm nhiều kịch bản cuộc gọi khác nhau. Nó có thể được mô tả bằng ví dụ chẳng hạn :

- Cuộc gọi giữa hai đối tác.
- Nhiều cuộc gọi đồng loạt xảy ra trên cùng thiết bị đầu cuối.
- Một cuộc hội thảo nhiều đối tác.
- Cài đặt cuộc gọi để thông báo nhiều thiết bị đầu cuối.

Mô hình cuộc gọi mô tả việc gọi cũng như những thành phần tham gia cuộc gọi. Tất cả nó định nghĩa trong 5 lớp cơ sở. Hai lớp mô tả những thành phần tham gia cuộc gọi. Những đối tượng duy trì và độc lập của cuộc gọi:

- Một người sử dụng (*user*) được đại diện bởi một đối tượng địa chỉ (*Address*). Thuộc tính chính của đối tượng địa chỉ là định danh người sử dụng (*user identifier*).
- Một điện thoại đầu cuối được đại diện cho bởi đối tượng đầu cuối (*Terminal*). Thuộc tính chính của đối tượng thiết bị đầu cuối là địa chỉ của thiết bị đó.

Ba lớp khác mô tả một cuộc gọi. Những đối tượng thể hiện của các lớp này không duy trì mà được tạo ra động trong khi cuộc gọi xảy ra. Mỗi đối tượng bao gồm một trạng thái máy hữu hạn:

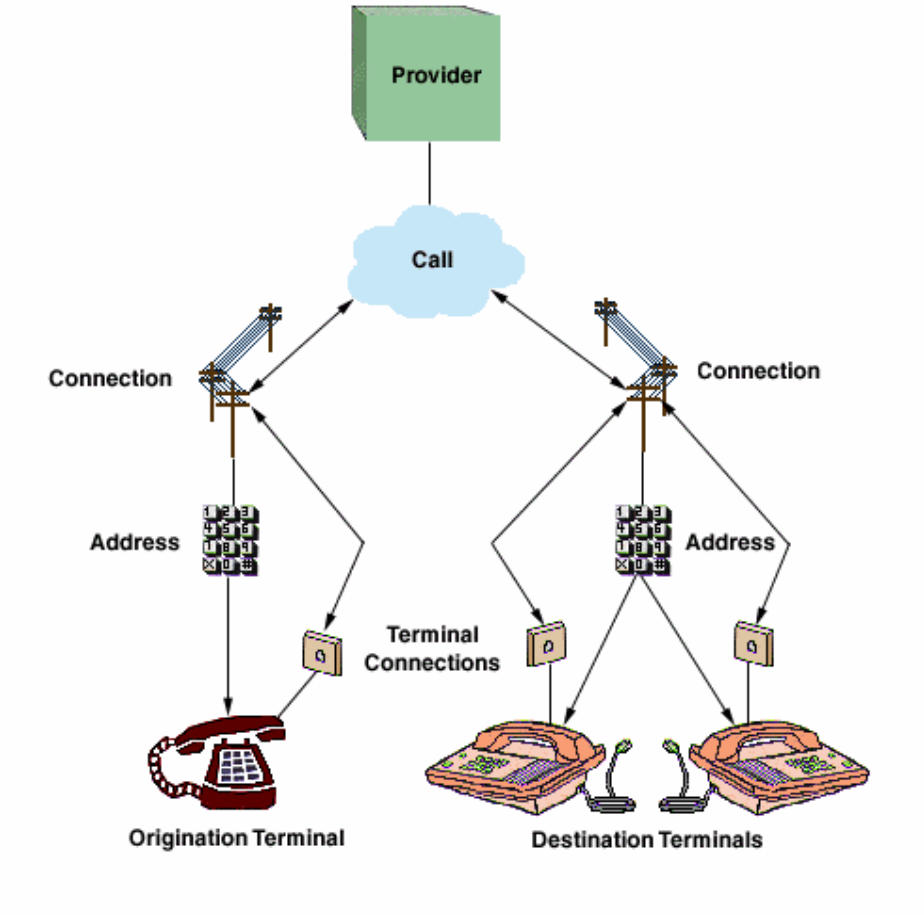
- Một đối tượng gọi (*Call*) được tạo ra cho mỗi cuộc gọi.
- Một đối tượng kết nối (*Connection*) được tạo ra cho mỗi người sử dụng tham gia vào cuộc gọi. Nó kết nối đối tượng địa chỉ của người sử dụng với đối tượng gọi.
- Một đối tượng kết nối đầu cuối (*TerminalConnection*) được tạo ra cho mỗi thiết bị đầu cuối tham gia vào cuộc gọi. Nó kết nối đối tượng (*Terminal*) thiết bị đầu cuối với đối tượng kết nối (*Connection*).

4.2. Các đối tượng trong mô hình gọi thoại java

Các đối tượng trong mô hình gọi thoại Java được thể hiện như hình 6.7.

- **Đối tượng Provider**: là một sự trừu tượng của phần mềm service-provider telephone. Provider có thể quản lý kết nối giữa PBX với server, một card telephony/fax trong máy desktop hoặc một công nghệ mạng máy tính như IP. Provider ẩn tất cả các chi tiết dịch vụ cụ thể của các hệ thống con telephone và cho phép ứng dụng Java hoặc Applet tương tác với các hệ thống con telephone trong cơ chế độc lập thiết bị.
- **Đối tượng Call**: Đối tượng này thể hiện một cuộc gọi điện thoại là luồng thông tin giữa người cung cấp dịch vụ và các thành viên của cuộc gọi. Một cuộc gọi điện thoại bao gồm một đối tượng Call và không hoặc nhiều kết nối. Trong kiểu gọi two-party gồm một đối tượng Call và 2 kết nối, còn trong kiểu hội thảo thì có 3 hoặc nhiều hơn số kết nối với một đối tượng Call.
- **Đối tượng Address**: Đối tượng này biểu diễn một số điện thoại. Nó là sự trừu tượng đối với một điểm cuối logic của một cuộc gọi điện thoại. Trong thực tế một số điện thoại có thể tương ứng với một số điểm cuối vật lý.
- **Đối tượng Connection**: Một đối tượng Connection mô hình hoá liên kết truyền thông giữa đối tượng Call và đối tượng Address. Đối tượng Connection có thể ở trong một

trong các trạng thái khác nhau chỉ thị trạng thái quan hệ hiện thời giữa Call và Address.



Hình 6.7. Mô hình cuộc gọi thoại Java

- **Đối tượng Terminal**: Biểu diễn một thiết bị vật lý như điện thoại và các thuộc tính gắn với nó. Mỗi đối tượng Terminal có một hoặc nhiều đối tượng Address(số điện thoại) gắn kết với nó. Terminal cũng được xem như là điểm cuối vật lý của một cuộc gọi vì nó tương ứng với một phần cứng vật lý.
- **Đối tượng TerminalConnection:** Thể hiện mối quan hệ giữa một kết nối và một điểm cuối vật lý của một cuộc gọi mà được biểu diễn bởi đối tượng Terminal. Đối tượng này mô tả trạng thái hiện thời của mối quan hệ giữa đối tượng Connection và Terminal cụ thể.

4.3. Các phương thức gói cốt lõi JTAPI

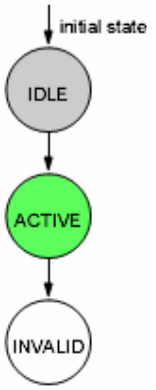
Gói cốt lõi của JTAPI định nghĩa 3 phương thức hỗ trợ các đặc trưng cơ bản: Thiết đặt một cuộc gọi, trả lời cuộc gọi và huỷ kết nối của một cuộc gọi. Các phương thức tương ứng với các tác vụ này là *Call.connect(), TerminalConnection.answer(), Connection.disconnect().*

- ***Phương thức Call.connect():***Khi một ứng dụng có đối tượng rỗi(thu được thông qua phương thức *Provider.createCall())*, nó có thể thiết lập một cuộc gọi điện thoại bằng cách sử dụng phương thức *Call.connect()*. Ứng dụng phải chỉ ra ddoois tượng Terminal nguồn(điểm cuối vật lý) và đối tượng Address nguồn(điểm cuối logic) trên Terminal đó. Nó cũng cung cấp một chuỗi số điện thoại đích. Hai đối tượng Connection được trả về từ phương thức *Call.connect()* biểu diễn các đầu cuối nguồn và đích của một cuộc gọi điện thoại.
- ***TerminalConnection.answer():*** Khi một cuộc gọi đi tới một Terminal, nó sẽ được chỉ thị bởi đối tượng TerminalConnection đối với Terminal đó trong trạng thái RINGING. Tại thời điểm đó, ứng dụng sẽ gọi phương thức *TerminalConnection.answer()* để trả lời cuộc gọi tới đó.
- ***Connection.disconnect():*** Phương thức này được gọi để loại bỏ Address từ một cuộc thoại. Đối tượng Connection biểu diễn quan hệ đối tượng Address với cuộc gọi điện thoại. Ứng dụng sẽ gọi phương thức này khi đối tượng Connection đang ở trạng thái CONNECTED và trả về kết quả là đối tượng Connetction chuyển đến trạng thái DISCONNECTED.

4.4. Những trạng thái máy hữu hạn

4.4.1.Đối tượng cuộc gọi

Mỗi đối tượng cuộc gọi được tạo ra mỗi khi thực hiện cuộc gọi. Trạng thái của đối tượng cuộc gọi phụ thuộc vào mã số của đối tượng kết nối và nó gồm các trạng thái thể hiện như hình 6.8.



Hình 6.8. Đối tượng gọi.

Trạng thái nhàn rỗi(IDLE): Đây là trạng thái khởi đầu cho mọi cuộc gọi. Trong trạng thái này, cuộc gọi không có kết nối nào.

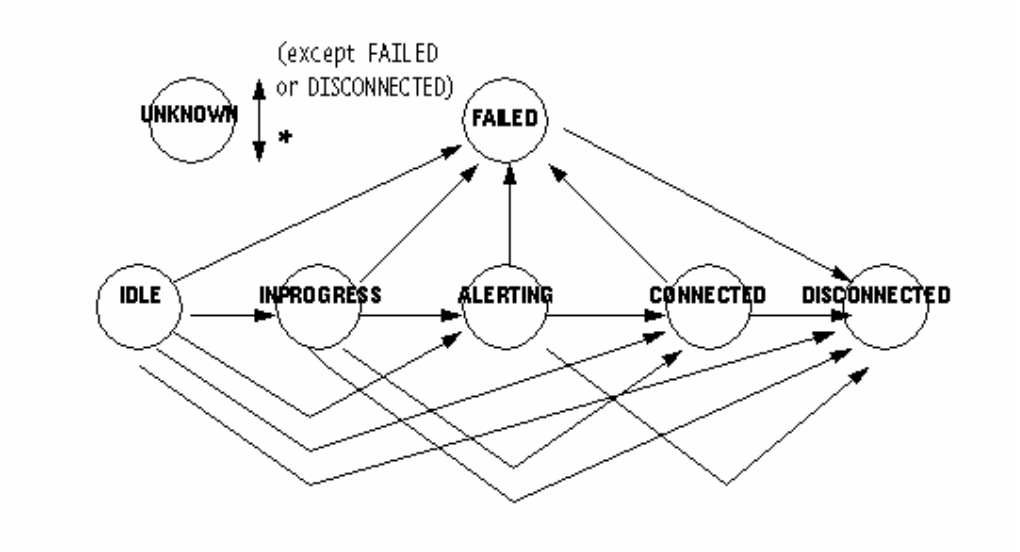
*Hoạt động(Active):*Đây trạng thái khi một cuộc gọi đang xẩy ra.Các cuộc gọi với một hoặc nhiều kết nối đều phải ở trong trạng thái này.

*Vô hiệu hóa(Invalid):*Đây là trạng thái cuối cùng cho mọi cuộc gọi. Cuộc gọi mà mất tất cả các đối tượng kết nối (thông qua một sự chuyển tiếp của đối tượng kết nối vào trong kết nối - trạng thái ngưng kết nối) sẽ chuyển vào trong trạng thái này. Các cuộc gọi khi ở trong trạng thái này sẽ không có kết nối nào và những đối tượng cuộc gọi này có thể không được sử dụng cho bất kỳ hoạt động nào trong tương lai. *2.4.1. Các trạng thái đối tượng Connection*

4.4.2. Các trạng thái đối tượng Connection và đối tượng TerminalConnection

Lược đồ dịch chuyển trạng thái của đối tượng Connection có thể được biểu diễn như hình 6.9. Nó gồm các trạng thái sau:

- **IDLE:** Đây là trạng thái khởi tạo ban đầu của tất cả các đối tượng Connection mới.
- **INPROGRESS:** Chỉ thị cuộc gọi điện thoại hiện thợi đã thiết đặt tới điểm cuối đích.
- **ALERTING:** Chỉ thị phía đích của cuộc gọi đã cảnh báo một cuộc gọi tới.
- **CONNECTED:** Chỉ thị trạng thái đurwcj kết nối của một cuộc điện thoại
- **DISCONNECTED:** Chỉ thị trạng thái kết thúc cuộc gọi.
- **FAILED:** Chỉ thị một cuộc gọi thiết đặt tới điểm cuối bị lỗi, ví dụ kết nối tới một phía đang bận.
- **UNKNOWN:** Chỉ thị rằng đối tượng Provider không thể xác định được đối tượng Connection tại thời điểm hiện thời.

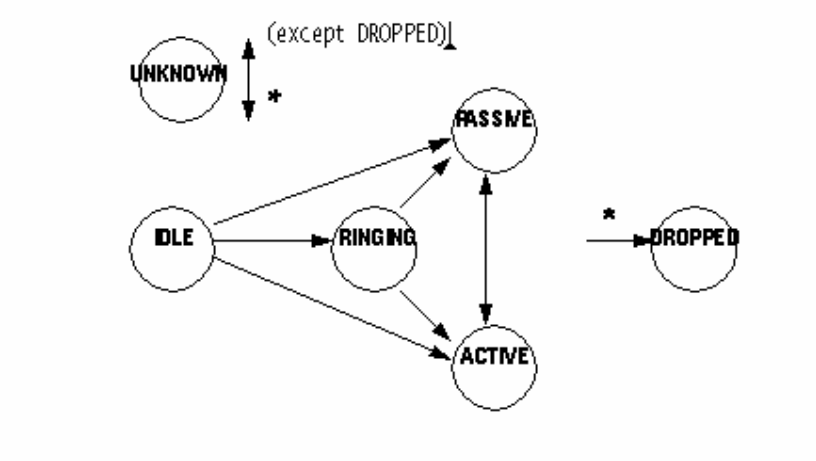


Hình 6.9. Lược đồ dịch chuyển trạng thái của Connection

4.4.3. *Các trạng thái đối tượng TerminalConnection*

Lược đồ dịch chuyển trạng thái của đối tượng TerminalConnection thể hiện như hình 6.10.

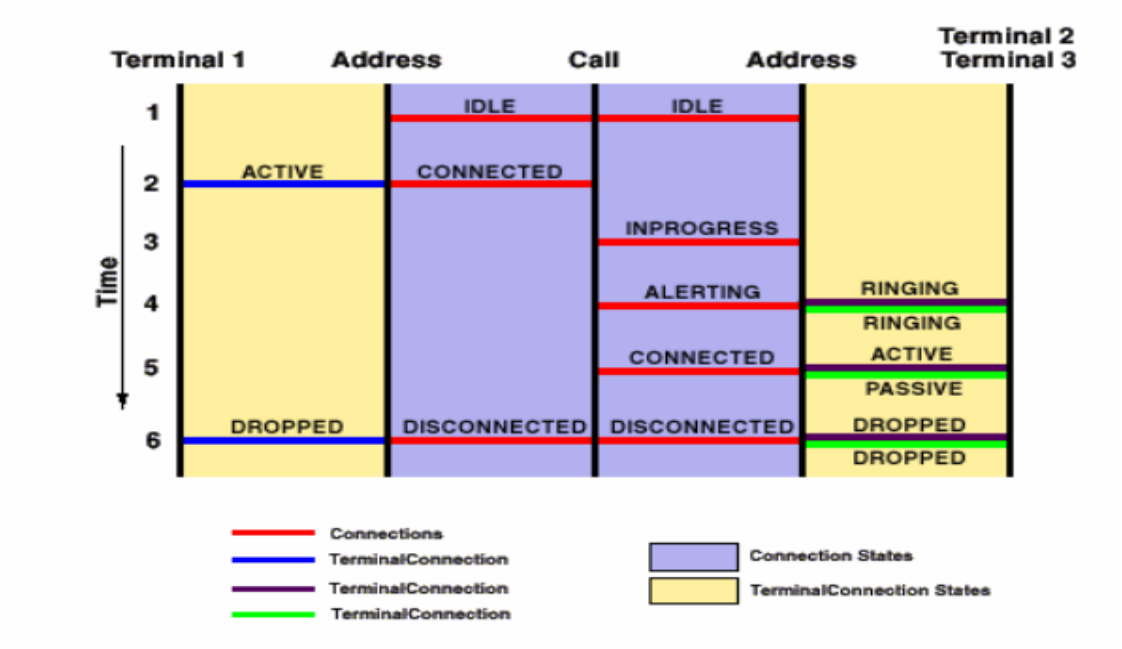
- **IDLE:** Trạng thái khởi tạo ban đầu của mọi đối tượng TerminalConnection
- **ACTIVE:** Chỉ thị Terminal là phần kích hoạt của một cuộc gọi điện thoại.
- **RINGING:** Chỉ thị rằng một Terminal báo tín hiệu cho người sử dụng có cuộc gọi tới tại Terminal hiện thời.
- **DROPPED:** Chỉ thị trạng thái bị dớt cuộc gọi
- **PASSIVE:** Chỉ thị trạng thái không kích hoạt của Terminal.
- **UNKNOWN:** Chỉ thị provider không cho phép xác định trạng thái hiện thời của TerminalConnection.



Hình 6.10. Lược đồ dịch chuyển trạng thái của TerminalConnection

4.5. Thiết đặt một cuộc gọi điện thoại

Phần này sẽ mô tả sự thay đổi trạng thái của toàn bộ mô hình gọi phải trải qua khi thiết đặt một cuộc gọi điện thoại đơn giản. Quá trình này có thể được thể hiện bằng một lược đồ định thời mô hình gọi như hình 6.11.



Hình 6.11. Lược đồ định thời mô hình cuộc gọi

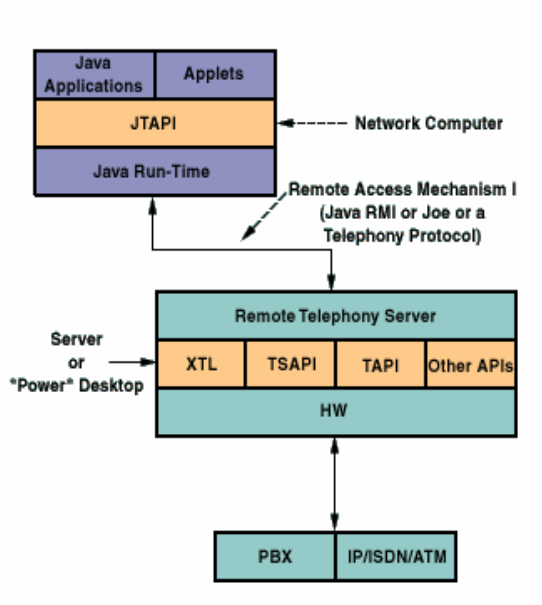
Trong lược đồ trên, các bước thời gian rời rạc bởi các số nguyên hướng xuống theo trục tung. Lược đồ này biểu diễn một cuộc gọi đơn giản kiểu two-party. Lược đồ này chia làm 2 phần, nửa trái và nửa phải. Nửa trái biểu diễn điểm cuối nguồn của cuộc gọi và nửa phải biểu diễn điểm cuối đích của cuộc gọi.

II. CẤU HÌNH HỆ THỐNG

JTAPI chạy trên nhiều cấu hình hệ thống khác nhau, bao gồm trung tâm phục vụ và máy tính mạng từ xa truy nhập tài nguyên điện thoại qua mạng. Trong cấu hình đầu tiên, một máy tính mạng đang chạy ứng dụng JTAPI và đang truy nhập những tài nguyên điện thoại qua một mạng được minh họa trong hình 6.12. Cấu hình thứ hai ứng dụng đang chạy trên một máy tính với những tài nguyên điện thoại riêng được minh họa trong hình 6.13.

1. Cấu hình máy tính mạng

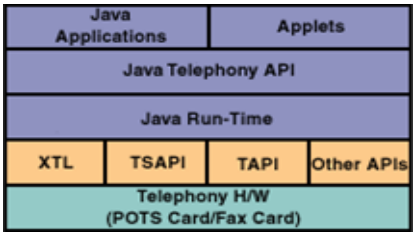
Trong cấu hình này, ứng dụng JTAPI hay Java applet chạy trên một trạm từ xa. Trạm làm việc này có thể là một máy tính nối mạng. Nó truy nhập tài nguyên mạng, sử dụng một trung tâm quản lý kỹ thuật điện thoại. JTAPI giao tiếp với bộ phận phục vụ này qua một cơ chế truyền thông từ xa, như RMI của Java, JOE hay một giao thức điện thoại nào đó. Cấu hình này đưwcj thể hiện như hình 6.10.



Hình 6.12. Cấu hình máy tính mạng

2. Cấu hình desktop

Trong cấu hình máy để bàn, ứng dụng JTAPI hay Java applet chạy trên cùng trạm làm việc. Cấu hình desktop thể hiện như hình 6.13.



Hình 6.13: Cấu hình Máy để bàn

III. MỘT SỐ VÍ DỤ LẬP TRÌNH VỚI JTAPI

1. Ví dụ thiết lập một cuộc gọi điện thoại sử dụng phương thức Call.connect()

```
import javax.telephony.*;
import javax.telephony.events.*;
/*
 * The MyOutCallObserver class implements the CallObserver
 * interface and receives all events associated with the Call.
 */

public class MyOutCallObserver implements CallObserver {

  public void callChangedEvent(CallEv[] evlist) {

    for (int i = 0; i < evlist.length; i++) {

      if (evlist[i] instanceof ConnEv) {

        String name = null;
        try {
          Connection connection = ((ConnEv)evlist[i]).getConnection();
          Address addr = connection.getAddress();
          name = addr.getName();
        } catch (Exception excp) {
          // Handle Exceptions
        }
        String msg = "Connection to Address: " + name + " is ";

        if (evlist[i].getID() == ConnAlertingEv.ID) {
          System.out.println(msg + "ALERTING");
        }
        else if (evlist[i].getID() == ConnInProgressEv.ID) {
          System.out.println(msg + "INPROGRESS");
        }
        else if (evlist[i].getID() == ConnConnectedEv.ID) {
          System.out.println(msg + "CONNECTED");
        }
        else if (evlist[i].getID() == ConnDisconnectedEv.ID) {
          System.out.println(msg + "DISCONNECTED");
        }
      }
    }
  }
}
```

2. Thực hiện cuộc gọi điện thoại từ một số tới một số

```
import javax.telephony.*;
import javax.telephony.events.*;
import MyOutCallObserver;
public class Outcall {
  public static final void main(String args[]) {
     /*
     * Create a provider by first obtaining the default implementation of
     * JTAPI and then the default provider of that implementation.
     */
```

```
    Provider myprovider = null;
    try {
      JtapiPeer peer = JtapiPeerFactory.getJtapiPeer(null);
      myprovider = peer.getProvider(null);
    } catch (Exception excp) {
      System.out.println("Can't get Provider: " + excp.toString());
      System.exit(0);
    }
   /*
    * We need to get the appropriate objects associated with the
    * originating side of the telephone call. We ask the Address for a list
    * of Terminals on it and arbitrarily choose one.
    */
    Address origaddr = null;
    Terminal origterm = null;
    try {
      origaddr = myprovider.getAddress("4761111");
      /* Just get some Terminal on this Address */
      Terminal[] terminals = origaddr.getTerminals();
      if (terminals == null) {
        System.out.println("No Terminals on Address.");
        System.exit(0);
      }
      origterm = terminals[0];
    } catch (Exception excp) {
      // Handle exceptions;
    }
    /*
     * Create the telephone call object and add an observer.
     */
    Call mycall = null;
    try {
      mycall = myprovider.createCall();
      mycall.addObserver(new MyOutCallObserver());
    } catch (Exception excp) {
      // Handle exceptions
    }
    /*
     * Place the telephone call.
     */
    try {
      Connection c[] = mycall.connect(origterm, origaddr, "5551212");
    } catch (Exception excp) {
      // Handle all Exceptions
    }
  }
}
```

3. Ví dụ minh hoạ cuộc gọi điện thoại tới

```
import javax.telephony.*;
import javax.telephony.events.*;
import javax.telephony.*;
```

```
import javax.telephony.events.*;
/*
 * The MyInCallObserver class implements the CallObserver and
 * recieves all Call-related events.
 */
public class MyInCallObserver implements CallObserver {

  public void callChangedEvent(CallEv[] evlist) {
    TerminalConnection termconn;
    String name;
    for (int i = 0; i < evlist.length; i++) {

      if (evlist[i] instanceof TermConnEv) {
        termconn = null;
        name = null;

        try {
          TermConnEv tcev = (TermConnEv)evlist[i];
          Terminal term = termconn.getTerminal();
          termconn = tcev.getTerminalConnection();
          name = term.getName();
        } catch (Exception excp) {
          // Handle exceptions.
        }

        String msg = "TerminalConnection to Terminal: " + name + " is ";

        if (evlist[i].getID() == TermConnActiveEv.ID) {
          System.out.println(msg + "ACTIVE");
        }
        else if (evlist[i].getID() == TermConnRingingEv.ID) {
          System.out.println(msg + "RINGING");

          /* Answer the telephone Call using "inner class" thread */
          try {
               final TerminalConnection _tc = termconn;
               Runnable r = new Runnable() {
                 public void run(){
                   try{
                              _tc.answer();
                   } catch (Exception excp){
                     // handle answer exceptions
                   }
                       };

               };
               Thread T = new Thread(r);
               T.start();
          } catch (Exception excp) {
            // Handle Exceptions;
          }
        } else if (evlist[i].getID() == TermConnDroppedEv.ID) {
          System.out.println(msg + "DROPPED");
```

```
        }
      }
    }
  }
}
import javax.telephony.*;
import javax.telephony.events.*;
import MyInCallObserver;

/*
 * Create a provider and monitor a particular terminal for an incoming
call.
 */
public class Incall {

  public static final void main(String args[]) {

    /*
     * Create a provider by first obtaining the default implementation of
     * JTAPI and then the default provider of that implementation.
     */
    Provider myprovider = null;
    try {
      JtapiPeer peer = JtapiPeerFactory.getJtapiPeer(null);
      myprovider = peer.getProvider(null);
    } catch (Exception excp) {
      System.out.println("Can't get Provider: " + excp.toString());
      System.exit(0);
    }

    /*
     * Get the terminal we wish to monitor and add a call observer to that
     * Terminal. This will place a call observer on all call which come to
     * that terminal. We are assuming that Terminals are named after some
     * primary telephone number on them.
     */
    try {
      Terminal terminal = myprovider.getTerminal("4761111");
      terminal.addCallObserver(new MyInCallObserver());
    } catch (Exception excp) {
      System.out.println("Can't get Terminal: " + excp.toString());
      System.exit(0);
  }
  }
}
```

4. Ví dụ xây dựng dịch vụ RAS với thư viện JTAPI

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import com.jpackages.jdun.*;
import javax.swing.border.*;
public class Do_an extends JFrame {
  public class DialNotify implements DialingNotification {
    // Phuong thuc goi lai
    public void notifyDialingState(String name, int state, int error) {
      // Hien thi y nghia cua trang thai quay so moi
```

```
      System.out.println("Tien trinh - " + name + ": (" + state + ") " +
DialingState.getDialingStateString(state));
      // Neu co loi thi hien thi y nghia cua ma loi
      if (error != 0) {
        System.out.println("Loi:"+error+""+
dum.getErrorMessageForCode(error));
      }
    }
  }
  // handle cho minh hoa ve quan ly quay so (DialUpManager)
  DialUpManager dum;
  // Minh hoa lop DialNotify (da dinh nghia o tren) ma co phuong thuc goi
lai
  DialNotify dnot = new DialNotify();
  // Dinh nghia giao dien do hoa
  JPanel contentPane;
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JScrollPane jScrollPane1 = new JScrollPane();
  DefaultListModel lm = new DefaultListModel();
  JList jList1 = new JList(lm);
  JPanel jPanel3 = new JPanel();
  BorderLayout borderLayout2 = new BorderLayout();
  JButton jButtonConnect = new JButton();
  JButton jButtonDisconnect = new JButton();
  BorderLayout borderLayout3 = new BorderLayout();
  JPanel jPanel4 = new JPanel();
  JLabel jLabel1 = new JLabel();
  JButton jButtonRefresh = new JButton();
  JPanel jPanel5 = new JPanel();
  FlowLayout flowLayout1 = new FlowLayout();
  JButton jButtonDelete = new JButton();
  JButton jButtonRename = new JButton();
  JPanel jPanel6 = new JPanel();
  JCheckBox jCheckBox1 = new JCheckBox();
  BorderLayout borderLayout4 = new BorderLayout();
  JPanel jPanel9 = new JPanel();
  JPanel jPanel7 = new JPanel();
  JTextField jTextFieldUsername = new JTextField();
  JLabel jLabel2 = new JLabel();
  JPasswordField jPasswordField1 = new JPasswordField();
  JPanel jPanel8 = new JPanel();
  JLabel jLabel3 = new JLabel();
  BorderLayout borderLayout5 = new BorderLayout();
  BorderLayout borderLayout6 = new BorderLayout();
  BorderLayout borderLayout7 = new BorderLayout();
  JPanel jPanel10 = new JPanel();
  JRadioButton jRadioButtonOverride = new JRadioButton();
  JRadioButton jRadioButtonDefault = new JRadioButton();
  JTextField jTextFieldPhoneNumber = new JTextField();
  JLabel jLabel4 = new JLabel();
  //Constructor
```

```
public Do_an() {
  try {
    // minh hoa lop quan ly quay so (DialUpManager)
    dum = new DialUpManager(dnot);
  }
  catch (LibraryLoadFailedException e) {
    if (e instanceof JDUNLibraryLoadFailedException)
      System.out.println("Khong the tai duoc thu vien JDUN...");
    else if (e instanceof RASLibraryLoadFailedException)
      System.out.println("Khong the tai duoc thu vien RAS... ");
    System.exit(0);
  }
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
    jbInit();
    initialize();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}
// kich hoat kha nang ghi de username/password
public void enableParams(boolean b) {
  this.jTextFieldUsername.setEnabled(b);
  this.jTextFieldUsername.setEditable(b);
  this.jPasswordField1.setEnabled(b);
  this.jPasswordField1.setEditable(b);
}
// khoi tao he thong
public void initialize() {
  this.enableParams(false);
  System.out.println("Dang khoi tao...");
  ButtonGroup bg = new ButtonGroup();
  bg.add(this.jRadioButtonDefault);
  bg.add(this.jRadioButtonOverride);
  this.refreshList();
}
// Danh sach JList
public void refreshList() {
  lm.clear();
  try {
    // Tim nap ten
    String[] names = dum.getEntryNames();
    for (int i=0; i < names.length; i++) {
      lm.addElement(names[i]);
    }
    this.jList1.repaint();
  }
  catch (Exception e) {}
}
//Khoi tao cac thanh phan
private void jbInit() throws Exception  {
  contentPane = (JPanel) this.getContentPane();
```

```
    contentPane.setLayout(borderLayout1);
    this.setSize(new Dimension(500, 400));
    this.setTitle("Chuong trinh minh hoa JDUNPhamHienTN2008@yahoo.com");
    jPanel1.setLayout(borderLayout2);
    jScrollPane1.setPreferredSize(new Dimension(660, 80));
    jPanel1.setPreferredSize(new Dimension(260, 100));
    jButtonConnect.setPreferredSize(new Dimension(105, 24));
    jButtonConnect.setText("Ket noi");
    jButtonConnect.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButtonConnect_actionPerformed(e);
      }
    });
    jButtonDisconnect.setPreferredSize(new Dimension(105, 24));
    jButtonDisconnect.setText("Ngat ket noi");
    jButtonDisconnect.addActionListener(new  java.awt.event.ActionListener()
{
      public void actionPerformed(ActionEvent e) {
        jButtonDisconnect_actionPerformed(e);
      }
    });
    jPanel2.setLayout(borderLayout3);
    jLabel1.setText("Nhap ten quay so");
    jButtonRefresh.setPreferredSize(new Dimension(79, 24));
    jButtonRefresh.setText("Lam lai");
    jButtonRefresh.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButtonRefresh_actionPerformed(e);
      }
    });
    jPanel5.setPreferredSize(new Dimension(70, 28));
    jPanel5.setLayout(flowLayout1);
    flowLayout1.setVgap(2);
    jButtonDelete.setPreferredSize(new Dimension(60, 24));
    jButtonDelete.setMargin(new Insets(0, 0, 0, 0));
    jButtonDelete.setText("Xoa");
    jButtonDelete.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButtonDelete_actionPerformed(e);
      }
    });
    jButtonRename.setPreferredSize(new Dimension(70, 24));
    jButtonRename.setMargin(new Insets(0, 0, 0, 0));
    jButtonRename.setText("Doi ten");
    jButtonRename.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButtonRename_actionPerformed(e);
      }
    });
    jPanel6.setBorder(BorderFactory.createEtchedBorder());
    jPanel6.setLayout(borderLayout4);
    jCheckBox1.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
```

```
        jCheckBox1_actionPerformed(e);
      }
    });
    jList1.addListSelectionListener(new
javax.swing.event.ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        jList1_valueChanged(e);
      }
    });
    jTextFieldUsername.setPreferredSize(new Dimension(200, 24));
    jLabel2.setPreferredSize(new Dimension(65, 17));
    jLabel2.setText("Nguoi dung");
    jPasswordField1.setPreferredSize(new Dimension(200, 24));
    jLabel2.setPreferredSize(new Dimension(65, 17));
    jLabel2.setText("Mat khau");
    jPanel9.setLayout(borderLayout5);
    jPanel8.setLayout(borderLayout7);
    jPanel7.setLayout(borderLayout6);
    jRadioButtonOverride.setText("Ghi de");
    jRadioButtonOverride.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jRadioButtonOverride_actionPerformed(e);
      }
    });
    jRadioButtonDefault.setSelected(true);
    jRadioButtonDefault.setText("Mac dinh");
    jRadioButtonDefault.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jRadioButtonDefault_actionPerformed(e);
      }
    });
    jTextFieldPhoneNumber.setPreferredSize(new Dimension(120, 24));
    jTextFieldPhoneNumber.setEditable(false);
    jLabel3.setText("So dien thoai");
    contentPane.add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(jScrollPane1, BorderLayout.CENTER);
    jPanel1.add(jPanel3, BorderLayout.NORTH);
    jPanel2.add(jPanel4, BorderLayout.CENTER);
    jPanel3.add(jLabel1, null);
    jPanel3.add(jButtonRefresh, null);
    jPanel2.add(jPanel5, BorderLayout.SOUTH);
    jPanel5.add(jButtonDelete, null);
    jPanel5.add(jButtonRename, null);
    jPanel1.add(jPanel6, BorderLayout.SOUTH);
    jPanel6.add(jCheckBox1, BorderLayout.WEST);
    jPanel6.add(jPanel9, BorderLayout.CENTER);
    jPanel8.add(jLabel3, BorderLayout.WEST);
    jPanel8.add(jPasswordField1, BorderLayout.CENTER);
    jPanel9.add(jPanel8, BorderLayout.SOUTH);
    jPanel9.add(jPanel7, BorderLayout.NORTH);
    jPanel7.add(jLabel2, BorderLayout.WEST);
```

```
    jPanel7.add(jTextFieldUsername, BorderLayout.CENTER);
    jPanel6.add(jPanel10, BorderLayout.SOUTH);
    jPanel10.add(jLabel4, null);
    jPanel10.add(jRadioButtonDefault, null);
    jPanel10.add(jRadioButtonOverride, null);
    jPanel10.add(jTextFieldPhoneNumber, null);
    jScrollPane1.getViewport().add(jList1, null);
    contentPane.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(jButtonConnect, null);
    jPanel2.add(jButtonDisconnect, null);
  }
  //Kha nang ghi de nho the ta co the thoat khi cua so duoc dong
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      System.exit(0);
    }
  }
  // Cap nhat lai JList
  void jButtonRefresh_actionPerformed(ActionEvent e) {
    this.refreshList();
  }
  // Ket noi toi doi tuong da chon
  void jButtonConnect_actionPerformed(ActionEvent e) {
    String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    // Tim nap so dien thoai ghi de
    String phonenum = null;
    if (this.jRadioButtonOverride.isSelected()) {
      phonenum = this.jTextFieldPhoneNumber.getText();
    }
    if (!jCheckBox1.isSelected()) {
      // Quay so voi username/password mac dinh
      if (phonenum == null) {
        dum.dialEntryAsynchronous(entryName);
      }
      else {
        dum.dialEntryAsynchronous(entryName, phonenum);
      }
    }
    else {
      // Lay username/password ghi de va su dung chung de quay so
      String username = this.jTextFieldUsername.getText();
      String password = new String(this.jPasswordField1.getPassword());
      if (phonenum == null) {
        dum.dialEntryAsynchronous(entryName, username, password);
      }
      else {
       dum.dialEntryAsynchronous(entryName,username,password,"",phonenum);
      }
    }
  }
```

```
  // Ket thuc cuoc goi da chon
  void jButtonDisconnect_actionPerformed(ActionEvent e) {
    final String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    dum.hangUpEntry(entryName);
  }
  // Xoa doi tuong duoc chon
  void jButtonDelete_actionPerformed(ActionEvent e) {
    String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    // Xac nhan xoa
    Int    eply=JOptionPane.showConfirmDialog(this,"Ban    co    chac    chan
muonxoa"+entryName+"khong?"Chuy...",JOptionPane.YES_NO_OPTION,JOptionPane.P
LAIN_MESSAGE);
    if (reply == JOptionPane.NO_OPTION) {
      return;
    }
    // Da xac nhan vi the xoa doi tuong
    dum.deleteEntry(entryName);
    // Cap nhat danh sach(JList) sau khi doi tuong da duoc xoa
    this.refreshList();
  }
  // Doi ten doi tuong duoc chon
  void jButtonRename_actionPerformed(ActionEvent e) {
    String entryName = (String) this.jList1.getSelectedValue();
    if (entryName == null)
      return;
    // Doi ten moi
    String message = "Nhap ten moi '" + entryName + "'";
    String  newname  =  (String)  JOptionPane.showInputDialog(this,  message,
"Doi ten", JOptionPane.PLAIN_MESSAGE, null, null, entryName);
    if (newname == null)
      return;
    if (newname.equals(entryName))
      return;
    // DOi ten bat ky doi tuong nao sang ten moi
    dum.renameEntry(entryName, newname);
    // Cap nhat  lai danh sac (JList) sau khi doi tuong duoc chon da duoc
doi ten
    this.refreshList();
  }
  void jCheckBox1_actionPerformed(ActionEvent e) {
    if (jCheckBox1.isSelected())
      this.enableParams(true);
    else
      this.enableParams(false);
  }
  // Khi mot danh sach cac lua chon duoc tao ra thi nap username/password
thich hop.
  void jList1_valueChanged(ListSelectionEvent e) {
    String entryName = (String) this.jList1.getSelectedValue();
```

```
    if (entryName == null)
      return;
    // Tim nap username/password
    String password = dum.getPassword(entryName);
    String username = dum.getUsername(entryName);
    // Hien thi username/password
    this.jTextFieldUsername.setText(username);
    this.jPasswordField1.setText(password);
    // Tim nap cac thuoc tinh doi tuong
    DialUpEntryProperties props = dum.getDialUpEntryProperties(entryName);
    // Hien thi so dien thoai
    if (props.getUseCountryAndAreaCodes()) {
      String areacode = props.getAreaCode();
      String phonenum = props.getLocalPhoneNumber();
      this.jTextFieldPhoneNumber.setText(areacode + phonenum);
    }
    else {
      this.jTextFieldPhoneNumber.setText(props.getLocalPhoneNumber());
    }
  }
  void phoneButtonChange() {
    if (this.jRadioButtonDefault.isSelected()) {
      this.jTextFieldPhoneNumber.setEditable(false);
    }
    else {
      this.jTextFieldPhoneNumber.setEditable(true);
    }
  }
  void jRadioButtonDefault_actionPerformed(ActionEvent e) {
    phoneButtonChange();
  }
  void jRadioButtonOverride_actionPerformed(ActionEvent e) {
    phoneButtonChange();
  }
  //Phuong thuc chinh
  public static void main(String[] args) {
    Do_an frame = new Do_an();
    //Can giua cho cua so
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width     -frameSize.width)     /     2,
(screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
  }
}
```

V. KẾT LUẬN

Trong chương nay chúng ta đã khảo sát gói thư viện JTAPI và kỹ thuật lập trình với nó. Qua chương này sinh viên nắm được cấu trúc của thư viện JTAPI, các khái niệm, mô hình và cách cài đặt chương trình với các cuộc gọi điện thoại đơn giản. Trên cơ sở đó sinh viên có thể phát triển các chương trình ứng dụng thực tế như dịch vụ truy cập từ xa RAS, hội thảo trực tuyến và các công nghệ liên qua đến IP khác, nhất là các dịch vụ trên hệ thống điện thoại doanh nghiệp(ET: Enterprise Telephony).