Mục lục

CHƯƠNG I: GIỚI THIỆU TỔNG QUAN

1. Lý do chọn đề tài

Công nghệ thông tin ngày nay có vai trò rất quan trọng trong cuộc sống hàng ngày của chúng ta. Hiện nay có rất nhiều công nghệ mới phát triển song song với việc phát triển công nghệ thông tin như Bluetooth, Wireless, WAP, SOAP,… nhằm giúp công nghệ thông tin ngày càng thân thiết với người dùng hơn. Một trong những công nghệ góp phần không nhỏ trong việc kết nối con người với thông tin cũng như con người với con người là công nghệ di động.Với tốc độ phát triển hiện nay và những lợi ích to lớn của công nghệ di động, có thể thấy nó có ảnh hưởng rất lớn đến cuộc sống của con người. Không giống như trước đây những chiếc điện thoại chỉ có chức năng rất đơn giản là đàm thoại, điện thoại hiện nay còn có thêm rất nhiều chức năng, ứng dụng khác như: email, truy cập Internet, video, nghe nhạc, chơi game, … đồng thời với nó là sự phát triển vũ bão của các dịch vụ gia tăng trên điện thoại di động dựa trên công nghệ WAP và SOAP.

Em chọn đề tài là "Lập trình thiết bị di động trên J2ME" và viết một số ứng dụng đơn giản nhằm khai thác các tính năng của các thiết bị di động mà chủ yếu là điện thoại di động. Qua đó em sẽ cố gắng nắm bắt và ứng dụng được tốt các kỹ thuật lập trình trên thiết bị di động.

2. Mục tiêu

Khi thực hiện đề tài này, mục tiêu mà em mong muốn đạt được là:

Hiểu chi tiết về J2ME và ứng dụng của nó để lập trình trên các thiết bị di động.

Nắm được các kỹ thuật xử lý form, âm thanh, hình ảnh, và lưu trữ trên điện thoại di động

Ứng dụng các kết quả đạt được để xây dựng chương trình đơn giản, có các tiện ích phục vụ nhu cầu của người sử dụng điện thoại di động

Áp dụng thành công trên một số dòng máy điện thoại di động hỗ trợ Java của các hãng như Nokia, Sony, Samsung,…

3. Đối tượng nghiên cứu

Hiểu chi tiết về J2ME và ứng dụng của nó để lập trình trên các thiết bị di động.

Nắm được các kỹ thuật xử lý âm thanh, hình ảnh, và lưu trữ dữ liệu trên thiết bị di động

Ứng dụng các kết quả có được để xây dựng một ứng dụng thực tiễn trên thiết bị di động

Tìm hiểu các công nghệ nâng cao trên điện thoại di động như Bluetooth, WAP, SOAP. Tìm hiểu về nguyên lý hoạt động của các dịch vụ gia tăng trên điện thoại di động.

Nếu còn thời gian, tìm hiểu về ý tưởng lập trình phân tán trên thiết bị di động Đây là một ý tưởng mới hầu như chưa được áp dụng cho thiết bị di động.

4. Phạm vi nghiên cứu

Nghiên cứu chi tiết về công nghệ J2ME và các kỹ thuật lập trình trên điên thoại di động. Ứng dụng các kết quả nghiên cứu được để xây dựng một ứng dụng triển khai trên điện thoại di động. Vì thời gian có hạn cũng như khả năng tìm hiểu còn nhiều hạn chế nên em chỉ trình bày các kỹ thuật lập trình trên một số dòng điện thoại phổ biển của các hãng lớn như Nokia, Samsung, Sony Ericssion. Em sẽ cố gắng khai thác các thế mạnh về form, âm thanh, hình ảnh mà các nhà sản xuất đã cung cấp trên điện thoại di động của họ.

Do không có đủ thiết bị để nghiên cứu nên em chỉ có thể trình bày những kỹ thuật lập trình trên điện thoại di động và các thiết bị di động khác nói chung. Do đó trong đề tài này, cụm từ “thiết bị di động” được hiểu theo nghĩa là “điện thoại di động”.

5. Môi trường thực hiện

Hệ điều hành Windows XP

IDE: NetBeans 5.5, NetBeans Mobility Pack 5.5.1 (đi kèm cả WTK 2.5)

JDK 1.6.02

Sun Wireless Toolkit 2.2

6. Giới thiệu về Java và công nghệ J2ME

Giới thiệu về Java

Java là một công nghệ được hãng Sun Microsystems xây dựng từ cuối năm 1990 với cái tên Oak và hiện nay đang phát triển vượt bậc với sự đóng góp của hàng vạn lập trình viên trên thế giới. Ban đầu, Oak được kỹ sư James Gosling và các cộng sự xây dựng với mục đích lập trình cho các mặt hàng điện dân dụng với mục tiêu nhỏ gọn và tương thích được với nhiều loại thiết bị phần cứng khác nhau. Sau đó Oak được sử dụng trong nhiều dự án như dự án Xanh (Blue Project), dự án Phim theo yêu cầu (Video on demand Project). Sau một chuyến du lịch tới đảo Java của Indonesia, nhóm phát triển Oak đã đổi tên Oak thành Java.

Java mà tiền thân là Oak được xây dựng chủ yếu dựa trên bộ công cụ phát triển (Java Development Kit - JDK) như là bộ thư viện chuẩn trong đó chưa trình biên dịch, trình thông dịch, trình đóng gói, tài liệu,… Đây chính là nền tảng cho việc phát triển các ứng dụng Java. Hiện nay, cộng đồng Java trên thế giới mà đi đầu là hãng Sun Microsystems đã xây dựng nhiều nhánh mới cho Java như: JavaMail (thư điện tử), Java TAPI (viễn thông), Java3D (đồ họa 3 chiều), J2ME (ứng dụng cho thiết bị di động),…

Hiện nay Java có các phiên bản sau:

J2SE^{TM} *(Java 2 Platform, Standart Edition)*: Phiên bản chuẩn gồm bộ công cụ thông dụng dùng để chạy trên các máy PC hoặc các mạng máy tính nhỏ.

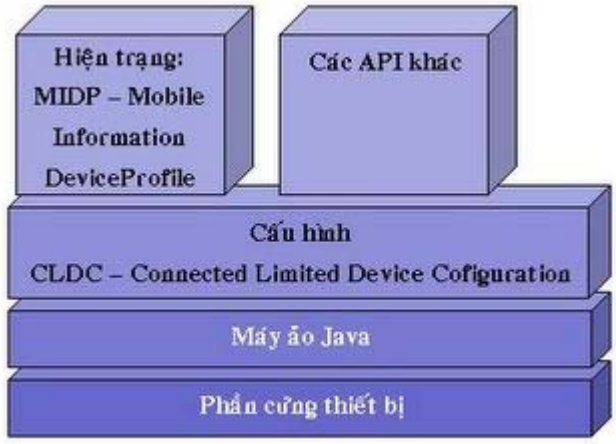
J2EE^{TM} (*Java 2 Platform, Enterprise Edition)*: Phiên bản dành cho các máy chủ với bộ nhớ lớn. Bao gồm các kiến trúc nâng cao như Web, EJB, Transaction,… dùng để xây dựng các ứng dụng có quy mô lớn

J2ME^{TM} *(Java 2 Platform, Micro Edition)*: Bao gồm môi trường và thư viện Java dùng để phát triển các ứng dụng trên các thiết bị có bộ nhớ nhỏ như điện thoại di động, PDA, các đồ gia dụng,…

Giới thiệu về J2ME và lập trình cho thiết bị di động

J2ME được phát triển từ kiến trúc JavaCard, EmbededJava và PersonalJava của phiên bản Java 1.1. Đến dự ra đời của phiên bản Java 2 thì Sun quyết định thay thế PersonalJava bằng một phiên bản mới có tên Java 2 Micro Edition, viết tắt là J2ME. J2ME được sử dụng cho các thiết bị nhỏ gọn với dung lượng bộ nhớ bé và khả năng xử lý thấp.

Mục tiêu của Java là cho phép người lập trình viết các ứng dụng độc lập với thiết bị di động, không cần quan tâm đến phần cứng thật sự. Để làm được như thế, J2ME được xây dựng bằng các tầng khác nhau để che giấu đi việc tương tác trực tiếp với phần cứng của thiết bị. Các tầng cảu J2ME được xây dựng trên CLDC (Connected Limited Device Configuration):



Tầng dưới cùng là tầng Phần cứng thiết bị - đây là tầng vật lý bao gồm phần cứng của thiết bị di động. Các tầng bên trên tầng Phần cứng thiết bị là các tầng trừu tượng, chúng cung cấp cho lập trình viên nhiều giao diện lập trình thân thiện và dễ dàng hơn mà không cần quan tâm đến phần cứng. Nói các khác chúng đóng vai trò trung gian giúp cho lập trình viên tương tác được với phần

cứng mà không cần quan tâm đến các chi tiết thực sự của phần cứng của thiết bị.

Tầng Phần cứng thiết bị (Device Hardware Layer): đây là thiết bị di động thật sự với bộ nhớ và tốc độ xử lý cụ thể. Các thiết bị di động khác nhau có thể có bộ vi xử lý và các tập lệnh rất khác nhau. Mục tiêu của J2ME là cung cấp cho lập trình viên khả năng giao tiếp giống nhau với tất cả các loại thiết bị di động khác nhau.

Tầng máy ảo Java (Java Virtual Machine Layer): đây là tầng đóng vai trò thông ngôn giữa chương trình và thiết bị. Nó sẽ thông dịch các mã bytecode (mã có được sau khi biên dịch mã nguồn chương trình) thành mã máy của các thiết bị di động. Tầng này bao gồm KVM (K Virtual Machine) là bộ biên dịch mã bytecode thành mã máy. Nó cung cấp một sự chuẩn hóa cho các thiết bị di động để ứng dụng J2ME sau khi biên dịch có thể chạy được trên bất kỳ thiết bị di động nào hỗ trợ KVM.

Tầng cấu hình (Configuration Layer): Tầng này cung cấp các hàm API cơ bản là nhân của J2ME. Lập trình viên có thể sử dụng các lớp và các phương thức của các API này tuy nhiên nó không thực sự phong phú bằng tập API của tầng hiện trạng.

Tầng hiện trạng (Profile Layer): Tầng này cung cấp các hàm API hữu dụng hơn cho việc lập trình. Mục đích của tầng này xây dựng nên lớp cấu hình và cung cấp nhiều thư viện ứng dụng hơn.

6.1 Lý do chọn J2ME:

Java ban đầu được thiết kế dành cho các máy với tài nguyên bộ nhớ hạn chế.

Thị trường của J2ME được mở rộng ra cho nhiều chủng loại thiết bị như:

Các loại thẻ cá nhân như Java Card

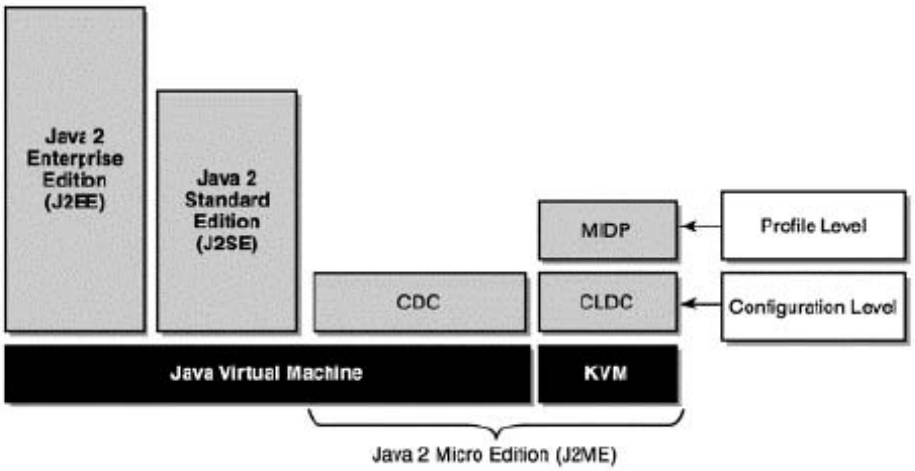
Máy điện thoại di động

Máy PDA (Personal Digital Assistant - thiết bị trợ giúp cá nhân)

Các hộp điều khiển dành cho tivi, thiết bị giải trí gia dụng …

6.2 Kiến trúc của J2ME

Phần này sẽ trình bày kiến trúc tổng quát của nền tảng Java



a) Giới thiệu các thành phần trong nền tảng J2ME: Định nghĩa về Configuration (Cấu hình): là đặc tảđịnh nghĩa một môi trường phần mềm cho một dòng các thiết bịđược phân loại bởi tập hợp các đặc tính, ví dụ như:

Kiểu và số lượng bộ nhớ

Kiểu và tốc độ bộ vi xử lý

Kiểu mạng kết nối

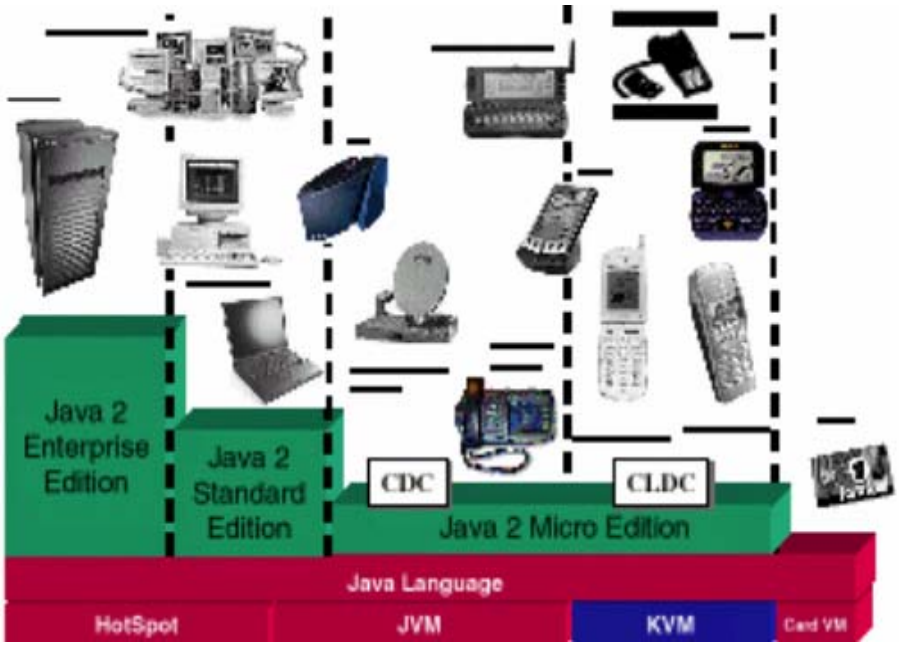
Do đây là đặc tả nên các nhà sản xuất thiết bị như Samsung, Nokia …bắt buộc phải thực thi đầy đủ các đặc tả do Sun qui định để các lập trình viên có thể dựa vào môi trường lập trình nhất quán và thông qua sự nhất quán này, các ứng dụng được tạo ra có thể mang tính độc lập thiết bịcao nhất có thể. Ví dụ như một lập trình viên viết chương trình game cho điện thoại Samsung thì có thể sửa đổi chương trình của mình một cách tối thiểu nhất để có thể chạy trên điện thọai Nokia.. Hiện nay Sun đã đưa ra 2 dạng Configuration:

CLDC (Connected Limited Device Configuration-Cấu hình thiết bị kết nối giới hạn): được thiết kếđể nhắm vào thị trường các thiết bị cấp thấp (low-end), các thiết bị này thông thường là máy điện thọai di động và PDA với khoảng 512 KB bộ nhớ. Vì tài nguyên bộ nhớ hạn chế nên CLDC được gắn với Java không dây (Java

Wireless), dạng như cho phép người sử dụng mua và tải về các ứng dụng Java, ví dụ như là Midlet.

CDC- Connected Device Configuration (Cấu hình thiết bị kết nối): CDC được đưa ra nhắm đến các thiết bị có tính năng mạnh hơn dòng thiết bị thuộc CLDC nhưng vẫn yếu hơn các hệ thống máy để bàn sử dụng J2SE. Những thiết bị này có nhiều bộ nhớ hơn (thông thường là trên 2Mb) và có bộ xử lý mạnh hơn. Các sản phẩm này có thể kểđến như các máy PDA cấp cao, điện thoại web, các thiết bị gia dụng trong gia đình …

Cả 2 dạng Cấu hình kể trên đều chứa máy ảo Java (Java Virtual Machine) và tập hợp các lớp (class) Java cơ bản để cung cấp một môi trường cho các ứng dụng J2ME. Tuy nhiên, bạn chú ý rằng đối với các thiết bị cấp thấp, do hạn chế về tài nguyên như bộ nhớ và bộ xử lý nên không thể yêu cầu máy ảo hổ trợ tất cả các tính năng như với máy ảo của J2SE, ví dụ, các thiết bị thuộc CLDC không có phần cứng yêu cầu các phép tính toán dấu phẩy động, nên máy ảo thuộc CLDC không được yêu cầu hỗ trợ kiểu float và double.



Bảng dưới là sự so sánh các thông số kỹ thuật của CDC và CLDC

	CLDC	CDC
Ram	>=32K, <=512K	>=256K
Rom	>=128k, <=512k	>=512k
Nguồn Năng Lượng	Có GiớI Hạn (nguồn pin)	Không giới hạn
Network	Chậm	Nhanh

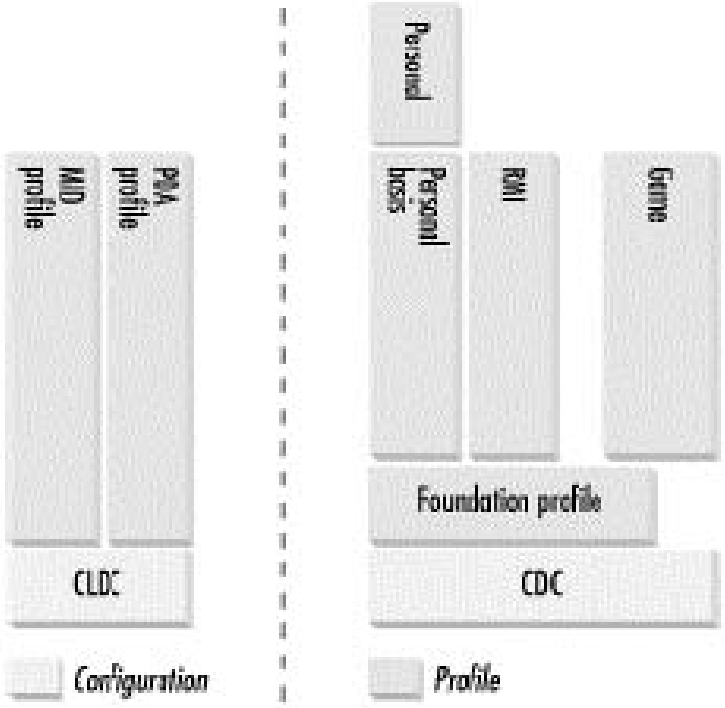
b) Định nghĩa về Profile:

Profile mở rộng Configuration bằng cách thêm vào các class để bổ trợ các tính năng cho từng thiết bị chuyên biệt. Cả 2 Configuration đều có những profile liên quan và từ những profile này có thể dùng các class lẫn nhau. Đến đây ta có thể nhận thấy do mỗi profile định nghĩa một tập hợp các class khác nhau, nên thường ta không thể chuyển một ứng dụng Java viết cho một profile này và chạy trên một máy hỗ trợ một profile khác. Cũng với lý do đó, bạn không thể lấy một ứng dụng viết trên J2SE hay J2EE và chạy trên các máy hỗtrợ J2ME. Sau đây là các profile tiêu biểu:

Mobile Information Device Profile (MIDP): profile này sẽ bổ sung các tính năng như hỗtrợ kết nối, các thành phần hỗ trợ giao diện người dùng … vào CLDC. Profile này được thiết kế chủ yếu để nhắm vào điện thoại di động với đặc tính là màn hình hiển thị hạn chế, dung lượng chứa có hạn. Do đó MIDP sẽ cung cấp một giao diện người dùng đơn giản và các tính năng mạng đơn giản dựa trên HTTP. Có thể nói MIDP là profile nổi tiếng nhất bởi vì nó là kiến thức cơ bản cho lập trình Java trên các máy di động (Wireless Java)

PDA Profile: tương tự MIDP, nhưng với thị trường là các máy PDA với màn hình và bộnhớ lớn hơn

Foundation Profile: cho phép mở rộng các tính năng của CDC với phần lớn các thư viện của bộ Core Java2 1.3 Ngoài ra còn có Personal Basis Profile, Personal Profile, RMI Profile, Game Profile.



6.3 Phát triển ứng dụng

Biên dịch

Mã nguồn chương trình có thể được biên dịch bằng các trình biên dịch chuẩn cảu Java, chúng tạo ra các file .class. Ta có thể biên dịch từ các trình soạn thảo hoặc biên dịch trực tiếp từ dòng lệnh.

6.4 Kiểm tra lỗi và chạy thử

Chúng ta sử dụng các công cụ như WTK để kiểm tra lỗi và chạy thử chương trình vì việc này nếu tiến hành trên thiết bị thật rất mất thời gian. Việc sử dụng các giả lập giúp nhanh chóng phát hiện các lỗi. Ngoài ra nó còn giúp lập trình viên có những cái nhìn cảm quan về chương trình của mình.

6.5 Đóng gói

Sau khi đã kiểm lỗi và chạy thử chương trình, chúng ta tiến hành đóng gói ứng dụng để có thể cài đặt trên các thiết bị thật. Việc đóng gói ứng dụng thực chất là nén các file .class vào trong một file .jar, điều này giúp giảm kích thước ứng dụng và đơn giản hóa khi cài đặt trên thiết bị thật. Chúng ta có thể đóng gói ứng dụng bằng trình đóng gói của JDK hoặc trình đóng gói nằm trong các IDE.

Hoặc một cách rất thủ công, chúng ta có thể đóng gói ứng dụng một cách trực tiếp. Việc đóng gói trực tiếp thực chất cũng tiến hành lại các công việc như các trình đóng gói nhưng chúng ta có thể kiểm soát lỗi tốt hơn. Tuy vậy việc này khá phức tạp và dễ gây ra lỗi nếu lập trình viên chưa thuần thục

6.6 Đóng gói và triển khai ứng dụng thành tập tin JAR

Các lớp đã được biên dịch của ứng dụng J2ME được đóng gói trong tập tin JAR cùng với các tài nguyên khác như hành ảnh, âm thanh,… Tập tin JAR này chính là tập tin được cài vào thiết bị di động.

Người sử dụng có thể tải tập tin JAR vào máy di động bằng các cách sau:

Kết nối điện thoại di động với máy tính bằng cáp truyền dữ liệu: Việc này yêu cầu người dùng phải có tập tin JAR thật sự và phần mềm truyền thông để tải ứng dụng vào điện thoại thông qua cáp dữ liệu

Cổng hồng ngoại: Yêu cầu thiết bị di động và nguồn chưa file JAR phải hỗ trợ hồng ngoại và người dùng có file JAR thật sự

Sử dụng mạng không dây: tải ứng dụng thông qua mạng GPRS, người dùng chỉ cần biết địa chỉ URL của tập tin JAR.

6.7 Tập tin manifest.mf và tập tin JAD

Tập tin manifest.mf và tập tin JAD mô tả các đặc điểm của ứng dụng. Tập tin manifest.mf nằm bên trong tập tin JAR còn tập tin JAD nằm ngoài tập tin JAR. Tập tin JAD giúp cho người dùng có thể biết được đặc điểm của ứng dụng trước

khi tải. Việc này giúp làm giảm lãng phí tài nguyên và tiền bạc vì trên thực tế, một ứng dụng J2ME nào đó chỉ có thể chạy trên một số máy nhất định.

Tập tin manifest.mf có nội dung như sau:

Manifest-Version: //Phiên bản tập tin manifest.mf

MIDlet-Name: //Tên bộ MIDlet

MIDlet-Version: //Phiên bản của bộ MIDlet

MIDlet-Vendor: //Nhà sản xuất

MIDlet-<n>: //Tên của MIDlet chính

MicroEdition-Profile: //Phiên bản hiện trạng

MicroEdition-Configuration: //Phiên bản cấu hình

6.8 Tối ưu mã chương trình và giảm kích thước ứng dụng

Sau khi đóng gói chương trình thành tập tin JAR chúng ta thấy rằng các file dữ liệu đã được nén lại một cách đáng kể. Tuy nhiên ta có thể giảm kích thước file JAR này thêm một lần nữa bằng cách dùng một công cụ. Công cụ này thường bao gồm các đặc tính sau:

- Loại bỏ các class không dùng đến
- Loại bỏ các hàm và biến không dùng đến
- Đổi tên class, package, hàm và biến thành các tên đơn giản và ngắn gọn hơn
- Thêm vào file class một số mã để chương trình khó bị dịch ngược hơn

Ba đặc tính đầu dùng để giảm kích thước các file .class trong khi đó đặc tính thứ 3 và thứ 4 dùng để bảo vệ chương trình khó bị dịch ngược lại thành mã nguồn. Ngay cả khi bị dịch ngược lại thành mã nguồn thì chương trình cũng khó bị đọc hơn vì các tên lớp, biến , hàm, package đã bị thay đổi. Các công cụ thường được dùng để tối ưu mã chương trình là Jbuilder 9X, Retroguard, Jshrink.

6.9 Những khó khăn khi lập trình trên thiết bị di động

Sử dụng công nghệ J2ME cho việc lập trình trên thiết bị di động là một việc không khó đối với các lập trình viên. Tuy vậy khi lập trình bằng J2ME, lập trình viên sẽ gặp phải một số khó khăn đạc trưng không thể tránh khỏi:

- Không hỗ trợ phép tính dấu phẩy động (floating point):
- Không hỗ trợ bộ nạp class (Class loader).
- Không hỗ trợ từ khóa finalize()
- Phần lớn các thư viện API cho Swing và AWT không thể sử dụng được trong MIDP.
- Không hỗ trợ các tính năng quản lý file và thư mục: Đây có thể làm bạn ngạc nhiên nhưng thực tế là các thiết bị J2ME không có hỗ trợ các thiết bị lưu trữ thông thường như ổ cứng v.v. Tuy nhiên, điều đó không có nghĩa là bạn phải mất đi mọi dữ liệu quan trọng mỗi khi tắt máy, Sun đã cung cấp một chức năng khác tương đương gọi là Record Management system (RMS) để cung cấp khả năng lưu trữ cho các thiết bị này.
- Các thiết bị di động bị giới hạn về kích thước ứng dụng. Ví dụ như với Series 40 của Nokia, Samsung X100, V200,… có dung lượng lưu trữ rất hạn chế. Sau đây là kích thước tối đa của file JAR cài đặt trên một số dòng điện thoại:

Loại điện thoại	Kích thước tối đa của file JAR
Nokia series 40	64 KB
Motorola T720	64KB
Panasonic X60	80KB
Sony Ericssion T610, T630	128KB
Samsung X600	100KB

Đó là một số khó khăn mà các lập trình viên thường gặp phải khi lập trình cho điện thoại di động. Trong giới hạn của đề tài này, em sẽ không đi cụ thể vào việc giải quyết các khó khăn này mà sẽ chủ yếu đi vào việc khác phục thông qua một số kỹ thuật khi tìm hiểu về các phần khác.

Chương II: LẬP TRÌNH VỚI J2ME

1. MIDlet và đối tượng Display

1.1 MIDlet – Vòng đời của một MIDlet

Nếu người nào đã viết Applet thì chắc hẳn thấy hai cái tên này na ná nhau. Thật vậy: MIDlet là viết tắt của "Mobile Information Device applet",.Hầu hết các ứng dụng mà ta thấy trên điện thoại di động đều là MIDlet.

Một MIDlet kế thừa từ lớp javax.microedition.midlet.MIDlet và thực thi ít nhất các phương thức cơ bản sau: startApp(), pauseApp(), và destroyApp(). Trong một ứng dụng của bạn gồm có nhiều lớp thì có thể chỉ cần một lớp kế thừa MIDlet. Ta sẽ đi vào phân tích từng đoạn nhỏ một trong đoạn code hoàn chỉnh của một MIDlet.

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class test extends MIDlet implements CommandListener{
        private Form mainForm;
        public test(){
                mainForm =   new     Form("Lap trinh tren nen J2ME");
                mainForm.append(new StringItem(null,"Hello J2ME"));
                mainForm.addCommand(new Command("Exit",Command.EXIT,0));
                mainForm.setCommandListener(this);
        }
        public void startApp(){
                Display.getDisplay(this).setCurrent(mainForm);
        }

        public void pauseApp(){}
        public void destroyApp(boolean un){}
        public void commandAction(Command c, Displayable s){
                notifyDestroyed();
```

```
        }
}
```

1) Phát biểu import: dùng để nạp các lớp cần thiết từ thư viện của CLDC và MIDP
2) Dòng khai báo lớp: một lớp(class) test có thể được gọi từ bất kỳ lớp khác (public), kế thừa (extends) từ lớp MIDlet (hay dễ hiểu hơn là: lớp test là một MIDlet) và gọi thực thi (implements) các phương thức của một interface có tên là CommandListener.
3) Hàm tạo (Constructor):

   Tạo ra một form có title là "Lap trinh tren nen J2ME"

   Gắn vào form vừa tạo một chuỗi là "Hello J2ME"

   Tạo ra một nút Exit trên form, tương tác tại nút 0, bạn thử thay 0 bằng 1,2 xem sao

   setCommandListener: Gắn sự kiện cho form

   Hàm tạo chỉ được gọi một lần khi MIDlet khởi tạo lần đầu tiên, và chỉ được gọi lại khi đã thoát ra khỏi MIDlet, rồi khởi động lại
4) startApp():

   Phương thức startApp() được gọi khi MIDlet được khởi tạo, và mỗi khi MIDlet trở về từ trạng thái tạm dừng (pause). Các biến toàn cục sẽ được khởi tạo lại trừ hàm tạo bởi vì các biến đã được giải phóng trong hàm pauseApp(). Nếu không thì chúng sẽ không được khởi tạo lại bởi ứng dụng.
5) pauseApp():

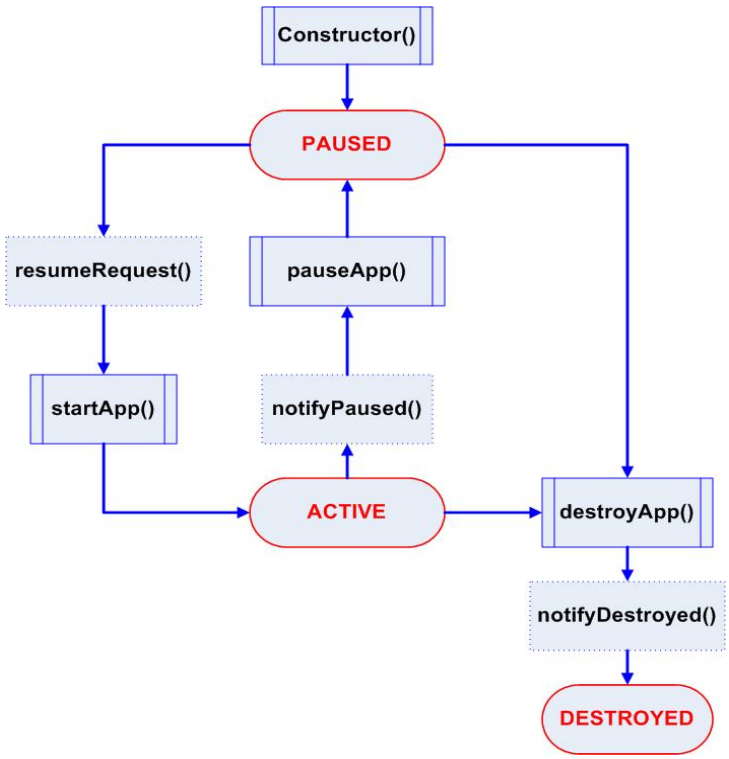
   Phương thức pauseApp() được gọi mỗi khi ứng dụng cần được tạm dừng (ví dụ, trong trường hợp có cuộc gọi hoặc tin nhắn đến). Cách thích hợp để sử dụng pauseApp() là giải phóng tài nguyên và các biến để dành cho các chức năng khác trong điện thoại trong khi MIDlet được tạm dừng. Cần chú ý rằng khi nhận cuộc gọi đến, hệ điều hành trên điện thoại di động có thể dừng

KVM thay vì dừng MIDlet. Việc này do nhà sản xuất thiết bị quyết định sẽ chọn cách nào.

6) destroyApp(boolean un):

Phương thức destroyApp() được gọi khi thoát MIDlet. (ví dụ khi nhấn nút exit trong ứng dụng). Nó chỉ đơn thuần là thoát MIDlet.. Phương thức destroyApp() chỉ nhận một tham số Boolean. Nếu tham số này là true, MIDlet được tắt vô điều kiện. Nếu tham số là false, MIDlet có thêm tùy chọn từ chối thoát bằng cách ném ra một ngoại lệ MIDletStateChangeException.

Dưới đây là vòng đời của một MIDlet:



Ngoại trừ các phương thức ta đã quen là startApp(), pauseApp(), destroyApp() chúng ta thấy có thêm 3 phương thức nữa, đó là: resumeRequest(), notifyPaused(), notifyDestroyed().

Từ sơ đồ khối trên, ta thấy:

MIDlet đang từ trạng thái PAUSED chuyển đến thực thi phương thức startApp() thông qua phương thức resumeRequest(): phương thức này yêu cầu MIDlet chuyển vào chế độ hoạt động.

MIDlet đang ở trạng thái hoạt động chuyển đến thực thi phương thức pauseApp() thông qua phương thức notifyPaused(): phương thức này cho biết MIDlet tự nguyện chuyển sang trạng thái dừng.

MIDlet đang ở trạng thái nào đó chuyển đến thực thi phương thức destroyApp() thông qua phương thức notifyDestroyed(): phương thức này cho biết MIDlet đã sắn sàng để hủy.

Từ đó chúng ta có thể thấy 3 phương thức mới này đặt MIDlet vào trạng thái trung gian giữa các trạng thái khác.

1.2 Đối tượng Display

Mỗi MIDlet có một tham chiếu đến một đối tượng Display. Đối tượng này cung cấp các thông tin về màn hình cũng như một số phương thức cần cho việc hiển thị các đối tượng khác trên màn hình. Có thể xem Display là đối tượng có nhiệm vụ quản lý việc hienẻ thị của màn hình. Chức năng của nó là quyết định danh sách các thành phần cần xuất hiện trên màn hình cũng như thời điểm phù hợp để hiển thị chúng.

1.3 Đối tượng Displayable

Mặc dù mỗi MIDlet chỉ có duy nhất một đối tượng Display nhưng nó lại có thể có rất nhiều đối tượng Displayable. Điều đó có nghĩa là một đối tượng Display có thể hiển thị bao nhiêu đối tượng Displayable tùy ý. Đối tượng Displayable là đối tượng có thể nhìn thấy được một cách trực quan trên màn hình. Bản thân MIDP có chứa 2 lớp con của Displayable là Screen và Canvas:

public abstract class Displayable

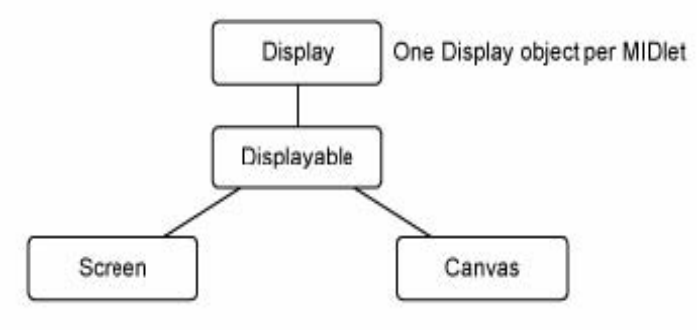
public abstract class Canvas extends Displayable

public abstract class Screen extends Displayable

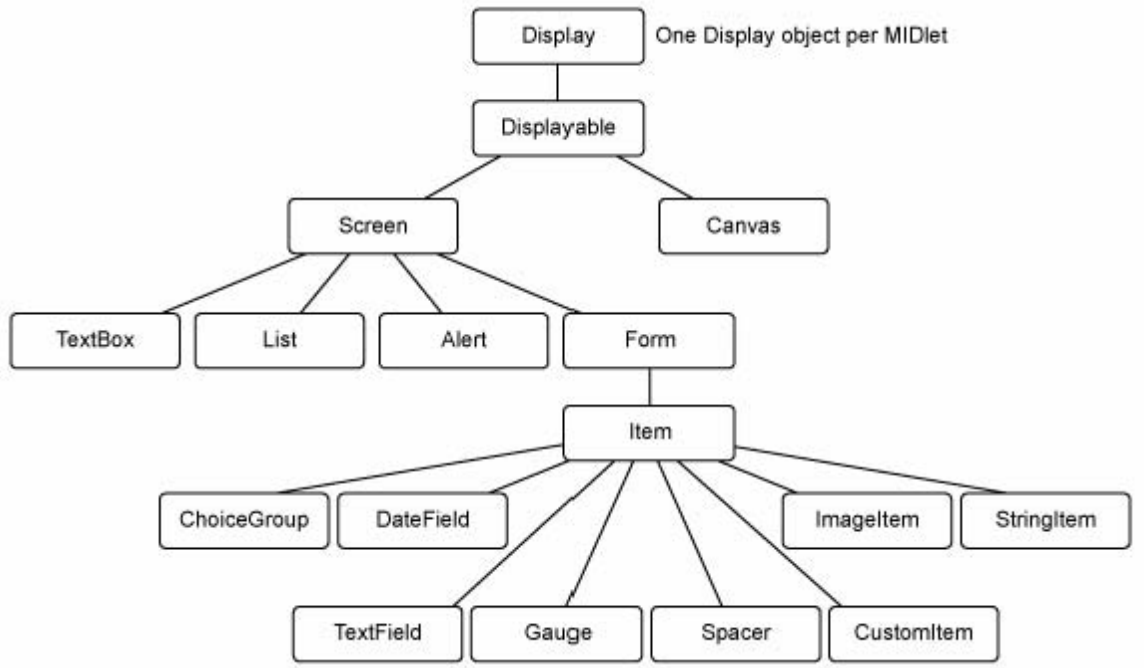
2. Giao diện người dùng cấp cao

2.1 Đối tượng Display, Displayable và Screens

Một ứng dụng MIDlet chỉ có 1 đối tượng thể hiện Display. Đối tượng này dùng để lấy thông tin vềđối tượng trình bày, ví dụ màu được hỗ trợ, và bao gồm các phương thức đểyêu cầu các đối tượng được trình bày. Đối tượng Display cần thiết cho bộ quản lý việc trình bày trên thiết bịđiều khiển thành phần nào sẽđược hiển thị lên trên thiết bịMặc dù chỉ có một đối tượng Display ứng với mỗi MIDlet, nhưng nhiều đối tượng trong một MIDlet có thểđược hiển thị ra trên thiết bị nhưForms, TextBoxes, ChoiceGroups, .. Một đối tượng Displayable là một thành phần được hiển thị trên một thiết bị. MIDP chứa 2 lớp con của lớp Displayable là Screen và Canvas. Hình dưới đây mô tả mối quan hệ trên



Một đối tượng Screen không phải là một cái gì đó hiện ra trên thiết bị, mà lớp Screen này sẽđược thừa kế bởi các thành phần hiển thịở mức cao, chính các thành phần này sẽđược hiển thị ra trên màn hình. Hình dưới đây sẽ mô tả mối quan hệ của lớp Screen và các thành phần thể hiện ở mức cao.



Tóm lại, phần này chỉ giới thiệu hệ thống phân cấp đối tượng dùng để thể hiện giao diện người dùng trong MIDP.

2.2 Thành phần Form và Items

Trong phần này sẽ giới thiệu các thành phần được hiển thị ra trên một Form. Một Form chỉ đơn giản là một khung chứa các thành phần, mà mỗi thành phần được thừa kế từ lớp Item. Chúng ta sẽ xem qua các thành phần hiển thị trên thiết bị:

DateField

Gauge

StringItem

TextField

ChoiceGroup

Spacer

CustomItem

Image and ImageItem

a) DateField

Thành phần DateField cung cấp một phương tiện trực quan để thao tác đối tượng Date được định nghĩa trong java.util.Date. Khi tạo một đối tượng DateField, bạn cần chỉ rõ là người dùng chỉ có thể chỉnh sửa ngày, chỉnh sửa giờ hay đồng thời cả hai. Các phương thức dựng của lớp DateField gồm:

DateField(String label, int mode)

DateField(String label, int mode, TimeZone timeZone)

Các mode tương ứng của lớp DateField gồm:

DateField.DATE_TIME: cho phép thay đổi ngày giờ

DateField.TIME: chỉ cho phép thay đổi giờ

DateField.DATE: chỉ cho phép thay đổi ngày

Ví dụ:

```
private DateField dfAlarm; // Tạo đối tượng DateField cho thay đổi cả ngày và giờ
dfAlarm = new DateField("Set Alarm Time", DateField.DATE_TIME);
dfAlarm.setDate(new Date());
```

Dưới đây là đoạn chương trình mẫu thử nghiệm đối tượng DateField

```
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.Timer;
import java.util.TimerTask;
public class DateFieldTest extends MIDlet implements ItemStateListener,
CommandListener {
        private Display display;       // Reference to display object
        private Form fmMain; // Main form
        private Command cmExit;  // Exit MIDlet
        private DateField dfAlarm;  // DateField component
        public DateFieldTest() {
                display = Display.getDisplay(this);
                // The main form
                 fmMain = new Form("DateField Test");
                // DateField with todays date as a default
                dfAlarm = new DateField("Set Alarm Time", DateField.DATE_TIME);
```

```
            dfAlarm.setDate(new Date());
            fmMain.addCommand(cmExit);
            fmMain.setCommandListener(this);
            fmMain.setItemStateListener(this);
        }
        public void startApp () {
            display.setCurrent(fmMain);
        }
        public void pauseApp() {}

        public void destroyApp(boolean unconditional) {}
        public void itemStateChanged(Item item) {
            System.out.println("Date field changed.");
        }
        public void commandAction(Command c, Displayable s) {
             if (c == cmExit) {
                 destroyApp(false); notifyDestroyed();
            }
        }
}
```

b) Gauge

Một thành phần Gauge là một kiểu giao diện thường được dùng để mô tả mức độhoàn thành một công việc. Có 2 loại Gauge là loại tương tác và loại không tương tác. Loại đầu cho phép người dùng có thể thay đổi Gauge, loại 2 thì đòi hỏi người phát triển phải cập nhật Gauge.

Dưới đây là hàm dựng của lớp Gauge:

Gauge(String label, boolean interactive, int maxValue, int initialValue)

Ví dụ:

private Gauge gaVolume; // Điều chỉnh âm lượng

gaVolume = new Gauge("Sound Level", true, 100, 4);

Dưới đây là đoạn chương trình mẫu minh họa cách sử dụng lớp Gauge

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class InteractiveGauge extends MIDlet implements CommandListener {
    private Display display; // Reference to display object
    private Form fmMain; // The main form
```

```
    private Command cmExit;  // Exit the form
    private Gauge gaVolume;  // Volume adjustment
    public InteractiveGauge() {
        display = Display.getDisplay(this);
        // Create the gauge and exit command
        gaVolume = new Gauge("Sound Level", true, 50, 4);
        cmExit = new Command("Exit", Command.EXIT, 1);
        // Create form, add commands, listen for events
        fmMain = new Form("");
        fmMain.addCommand(cmExit);
        fmMain.append(gaVolume);
        fmMain.setCommandListener(this);
    }
    // Called by application manager to start the MIDlet.
    public void startApp() {
        display.setCurrent(fmMain);
    }

    public void pauseApp() {}

    public void destroyApp(boolean unconditional) {}

    public void commandAction(Command c, Displayable s) {
        if (c == cmExit) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
```

c) StringItem

Một thành phần StringItem được dùng để hiển thị một nhãn hay chuỗi văn bản. Người dùng không thể thay đổi nhãn hay chuỗi văn bản khi chương trình đang chạy. StringItem không nhận ra sự kiện Phương thức dựng của lớp StringItem

StringItem(String label, String text)

Dưới đây là đoạn mã minh họa việc sử dụng đối tượng StringItem

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
```

```
public class StringItemTest extends MIDlet implements CommandListener{
    private Display display; // Reference to Display object
    private Form fmMain; // Main form private
    StringItem siMsg; // StringItem
    private Command cmChange;  // Change the label and message
    private Command cmExit;  // Exit the MIDlet
    public StringItemTest() {
        display = Display.getDisplay(this);
        // Create text message and commands
        siMsg = new StringItem("Website: ", "www.IBM.com");
        cmChange = new Command("Change", Command.SCREEN, 1);
        cmExit = new Command("Exit", Command.EXIT, 1);
        // Create Form, add Command and StringItem, listen for events
        fmMain = new Form("StringItem Test");
        fmMain.addCommand(cmExit);
         fmMain.addCommand(cmChange);
         fmMain.append(siMsg);
         fmMain.setCommandListener(this);
    }

    // Called by application manager to start the MIDlet.
    public void startApp() {
        display.setCurrent(fmMain);
    }
    public void pauseApp() {}
    public void destroyApp(boolean unconditional) {}

    public void commandAction(Command c, Displayable s) {
        if (c == cmChange) {
            // Change label
            siMsg.setLabel("Section: ");

            // Change text
            siMsg.setText("developerWorks");

            // Remove the command
            fmMain.removeCommand(cmChange);
        }
        else if (c == cmExit) {
            destroyApp(false);
            notifyDestroyed();
        }
```

```
    }
}
```

d) TextField

Một thành phần TextField thì tương tự như bất kỳ các đối tượng nhập văn bản tiêu biểu nào. Bạn có thể chỉ định một nhãn, số ký tự tối đa được phép nhập, và loại dữliệu được phép nhập. Ngoài ra TextField còn cho phép bạn nhập vào mật khẩu với các ký tự nhập vào sẽ được che bởi các ký tự mặt nạ

Phương thức dựng của lớp

TextField TextField(String label, String text, int maxSize, int constraints)

Thành phần thứ 3 constraints là thành phần mà chúng ta quan tâm, vì nó là phương tiện để xác định loại dữ liệu nào được phép nhập vào TextField. MIDP định nghĩa các tham số ràng buộc sau cho thành phần TextField:

ANY: cho phép nhập bất kỳ ký tự nào

EMAILADDR: chỉ cho phép nhập vào các địa chỉ email hợp lệ

NUMERIC: chỉ cho phép nhập số

PHONENUMBER: Chỉ cho phép nhập sốđiện thoại

URL: Chỉ cho phép nhập các ký tự hợp lệ bên trong URL

PASSWORD: che tất cả các ký tự nhập vào

Dưới đây là đoạn mã minh họa việc sử dụng thành phần TextField

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class TextFieldTest extends MIDlet implements CommandListener{
    private Display display; // Reference to Display object
    private Form fmMain; // Main form
    private Command cmTest;  // Get contents of textfield
    private Command cmExit;  // Command to exit the MIDlet
    private TextField tfText; // Textfield
    public TextFieldTest() {
        display = Display.getDisplay(this);
        // Create commands
        cmTest = new Command("Get Contents", Command.SCREEN, 1);
```

```
            cmExit = new Command("Exit", Command.EXIT, 1);
            // Textfield for phone number
            tfText = new TextField("Phone:", "", 10, TextField.PHONENUMBER);
            // Create Form, add Commands and textfield, listen for events
            fmMain = new Form("Phone Number");
            fmMain.addCommand(cmExit);
            fmMain.addCommand(cmTest);
             fmMain.append(tfText);
            fmMain.setCommandListener(this);
        }
        // Called by application manager to start the MIDlet.
        public void startApp() {
            display.setCurrent(fmMain);
        }
        public void pauseApp() {}
        public void destroyApp(boolean unconditional) {}

        public void commandAction(Command c, Displayable s) {
             if (c == cmTest) {
                System.out.println("TextField contains: " + tfText.getString());
            }
            else if (c == cmExit) {
                destroyApp(false);
                notifyDestroyed();
            }
        }
}
```

Đoạn mã trên chỉ mới áp dụng một ràng buộc trên đối tượng TextField. Chúng ta có thể thêm một ràng buộc thứ 2 bằng cách thay đoạn mã sau:

tfText = new TextField("Phone:", "", 10, TextField.PHONENUMBER | TextField.PASSWORD);

e) ChoiceGroup

Thành phần ChoiceGroup cho phép người dùng chọn từ một danh sách đầu vào đã được định nghĩa trước. ChoiceGroup có 2 loại:

multi-selection(cho phép chọn nhiều mục): nhóm này có liên quan đến các checkbox

exclusive-selection(chỉđược chọn một mục): nhóm này liên quan đến nhóm các radio button

Dưới đây là đoạn mã minh họa cho việc sử dụng ChoiceGroup:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ChoiceGroupTest extends MIDlet implements ItemStateListener,
CommandListener {
        private Display display;   // Reference to display object
        private Form fmMain; // Main form
        private Command cmExit;  // A Command to exit the MIDlet
        private Command cmView;  // View the choice selected
        private int selectAllIndex;  // Index of the "Select All" option
        private ChoiceGroup cgPrefs; // Choice Group of preferences
        private int choiceGroupIndex; // Index of choice group on form
        public ChoiceGroupTest() {
                 display = Display.getDisplay(this);
                // Create a multiple choice group
                cgPrefs = new ChoiceGroup("Preferences", Choice.MULTIPLE);
                 // Append options, with no associated images
                cgPrefs.append("Replace tabs with spaces", null);
                cgPrefs.append("Save bookmarks", null);
                cgPrefs.append("Detect file type", null);
                selectAllIndex = cgPrefs.append("Select All", null);
                cmExit = new Command("Exit", Command.EXIT, 1);
                cmView = new Command("View", Command.SCREEN,2);
                // Create Form, add components, listen for events
                fmMain = new Form("");
                choiceGroupIndex = fmMain.append(cgPrefs);
                fmMain.addCommand(cmExit);
                fmMain.addCommand(cmView);
                fmMain.setCommandListener(this);
                fmMain.setItemStateListener(this);
        }
        public void startApp() {
                display.setCurrent(fmMain);
        }
        public void pauseApp() {}
        public void destroyApp(boolean unconditional) {}
```

```
    public void commandAction(Command c, Displayable s) {
        if (c == cmView) {
            boolean selected[] = new boolean[cgPrefs.size()];
            // Fill array indicating whether each element is checked
            cgPrefs.getSelectedFlags(selected);
            for (int i = 0; i < cgPrefs.size(); i++)
            System.out.println(cgPrefs.getString(i) +   (selected[i] ? ":
            selected" : ": not selected"));
        }
        else if (c == cmExit) {
            destroyApp(false); notifyDestroyed();
        }
    }

    public void itemStateChanged(Item item) {
        if (item == cgPrefs) {
            // Is "Select all" option checked ?
            if (cgPrefs.isSelected(selectAllIndex)) {
                // Set all checkboxes to true
                for (int i = 0; i < cgPrefs.size(); i++)
                    cgPrefs.setSelectedIndex(i, true);
                // Remove the check by "Select All"
                cgPrefs.setSelectedIndex(selectAllIndex, false);
            }
        }
    }
}
```

f) Spacer

Spacer là thành phần không nhìn thấy, được dùng để định vị trí cho các đối tượng khác trên màn hình hiển thị. Chúng ta có thể dùng Spacer để chỉ rõ khoảng trắng theo chiều dọc và chiều ngang giữa các thành phần, đơn giản bằng cách chỉ ra chiều dài và chiều rộng cho từng cái. Vì Spacer là thành phần không nhìn thấy nên nó không có sự kiện

g) CustomItem

Thành phần CustomItem cho phép bạn tạo ra những thành phần Item của chính bạn. Những thành phần này cũng giống như những Item khác là cũng có thểđược đặt vào trong Form và có thể nhận biết và xử lý sự kiện

CustomItem được vẽ lên màn hình hiển thị bằng phương thức paint(). Vì thế nó sẽ tùy thuộc vào đoạn mã được bạn hiện thực bên trong phương thức paint(). Quá trình tạo ra một đối tượng CustomItem cũng không khác các đối tượng có sẵn trên nền Java.

Đoạn mã dưới đây minh họa sườn của việc tạo ra một đối tượng CustomItem

```
public class NewItem extends CustomItem {
        public NewItem(String label) {
                super(label);
                ...
        }
        protected void paint(Graphics g, int width, int height) {
                 ...
        }
        protected int getMinContentHeight() {
                 ...
        }
        protected int getMinContentWidth() {    ... }
        protected int getPrefContentHeight(int width) {     ...  }
        protected int getPrefContentWidth(int height) {
                ...
        }
        ...
}
```

h) Image and ImageItem

Hai lớp được dùng để hiển thị hình ảnh là: Image và ImageItem. Image được dùng để tạo ra một đối tượng hình ảnh và giữ thông tin như là chiều cao và chiều rộng, và dù ảnh có biến đổi hay không.

Lớp ImageItem mô tả một tấm ảnh sẽđược hiển thị như thế nào, ví dụ tấm ảnh sẽ được đặt ở trung tâm, hay đặt về phía bên trái, hay bên trên của màn hình.

MIDP đưa ra 2 loại hình ảnh là loại ảnh không biến đổi và ảnh biến đổi. Một tấm ảnh không biến đổi thì không thể bị thay đổi kể từ lúc nó được tạo ra. Đặc trưng của loại ảnh này là được đọc từ một tập tin. Một tấm ảnh biến đổi về cơ bản là một vùng nhớ. Điều này tùy thuộc vào việc bạn tạo nội dung của tấm ảnh bằng cách ghi nó lên vùng nhớ. Chúng ta sẽ làm việc với những tấm ảnh không biến đổi trong bảng sau.

Các phương thức dựng cho lớp Image và ImageItem

Image createImage(String name)

Image createImage(Image source)

Image createImage(byte[] imageDate, int imageOffset, int imageLength)

Image createImage(int width, int height)

Image createImage(Image image, int x, int y, int width, int height, int transform)

Image createImage(InputStream stream)

Image createRGBImage(int[] rgb, int width, int height, boolean processAlpha)

ImageItem(String label, Image img, int layout, String altText)

Đoạn mã dưới đây mô tả làm thế nào tạo một tấm ảnh từ một tập tin, và gắn nó với một đối tượng ImageItem và thêm một bức ảnh vào một Form

```
Form fmMain = new Form("Images");
 ... // Create an image
Image img = Image.createImage("/house.png");
// Append to a form
fmMain.append(new ImageItem(null, img, ImageItem.LAYOUT_CENTER, null));
```

Chú ý: PNG là loại ảnh duy nhất được hỗ trợ bởi bất kỳ thiết bị MIDP nào

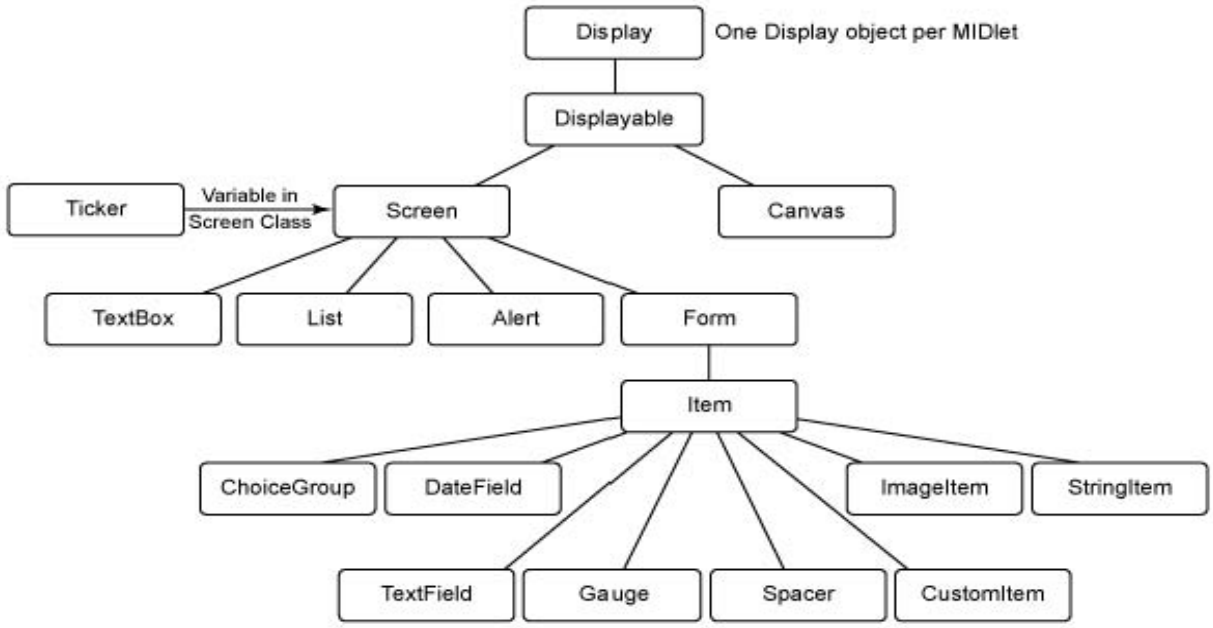
Đoạn mã dưới đây mô tả việc sử dụng đối tượng Image và đối tượng ImageItem

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ImageTest extends MIDlet  implements CommandListener {
        private Display display; // Reference to Display object
        private Form fmMain; // The main form
        private Command cmExit;  // Command to exit the MIDlet
        public ImageTest() {
```

```
        display = Display.getDisplay(this);
        cmExit = new Command("Exit", Command.EXIT, 1);
        fmMain = new Form("");
        fmMain.addCommand(cmExit);
        fmMain.setCommandListener(this);
        try {
                // Read the appropriate image based on color support
                Image im = Image.createImage((display.isColor()) ?

        "/image_color.png":"/image_bw.png");
                fmMain.append(new ImageItem(null, im,
                ImageItem.LAYOUT_CENTER, null));
        display.setCurrent(fmMain);
        }
        catch (java.io.IOException e) {
                System.err.println("Unable to locate or read .png file");
        }
    }

    public void startApp() {
        display.setCurrent(fmMain);
    }
    public void pauseApp() {}

    public void destroyApp(boolean unconditional) {}

    public void commandAction(Command c, Displayable s) {
        if (c == cmExit) {
                destroyApp(false); notifyDestroyed();
        }
    }
}
```

2.3 Thành phần List, Textbox, Alert, và Ticker

Trong phần này chúng ta sẽ xem xét các đối tượng ListBox, TextBox, Alert, và Ticker trong các thành phần giao diện cấp cao của ứng dụng MIDP. Chúng ta hãy cũng xem lại cây phân cấp các thành phần trình bày trên thiết bị một cách hoàn chỉnh hơn



a) List

Một List chứa một dãy các lựa chọn được thể hiện một trong ba dạng. Chúng ta đã thấy loại cho phép nhiều lựa chọn và loại chỉ được phép chọn một khi làm việc với ChoiceGroup. Dạng thứ 3 là là dạng không tường minh. Các List không tường minh được dùng để thể hiện một thực đơn các chọn lựa

Đoạn mã dưới đây minh họa việc sử dụng một danh sách không tường minh

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ImplicitList extends MIDlet implements CommandListener {
        private Display display; // Reference to Display object
        private List lsDocument;  // Main list
        private Command cmExit;  // Command to exit
        public ImplicitList() {
                display = Display.getDisplay(this);
                // Create the Commands
```

```
        cmExit = new Command("Exit", Command.EXIT, 1);
        try {
                // Create array of image objects
                Image images[] = {Image.createImage("/next.png");
                Image.createImage("/previous.png");
                Image.createImage("/new.png")};
                // Create array of corresponding string objects
                String options[] = {"Next", "Previous", "New"};
                 // Create list using arrays, add commands, listen for events
                lsDocument = new List("Document Option:", List.IMPLICIT,
                options, images);
                // If you have no images, use this line to create the list //
                lsDocument = new List("Document Option:", List.IMPLICIT,
                options, null);
                lsDocument.addCommand(cmExit);
                lsDocument.setCommandListener(this);
        }
        catch (java.io.IOException e) {
                System.err.println("Unable to locate or read .png file");
        }
}
public void startApp() {
        display.setCurrent(lsDocument);
}
public void pauseApp() { }
public void destroyApp(boolean unconditional) { }

public void commandAction(Command c, Displayable s) {
        // If an implicit list generated the event
        if (c == List.SELECT_COMMAND) {
                switch (lsDocument.getSelectedIndex())  {
                        case 0:
                         System.out.println("Next selected");
                         break;
                         case 1:
                        System.out.println("Previous selected");
                        break;
                        case 2:
                        System.out.println("New selected");
                        break;
                }
        }
```

```
            else if (c == cmExit) {
                destroyApp(false);
                notifyDestroyed();
            }
        }
}
```

b) TextBox

TextBox được dùng để cho phép nhập nhiều dòng. Thành phần TextBox và TextField có những ràng buộc giống nhau trong việc chỉđịnh loại nội dung được phép nhập vào. Ví dụ ANY, EMAIL, URI…

Dưới đây là phương thức dựng của một TextBox:

TextBox(String title, String text, int maxSize, int constraints)

c) Alert và AlertType

Một Alert đơn giản là một hộp thoại rất nhỏ. Có 2 loại Alert:

Modal: là loại hộp thoại thông báo được trình bày cho đến khi người dùng ấn nút đồng ý

Non-modal: là loại hộp thoại thông báo chỉđược trình bày trong một số giây nhất định

Các phương thức dựng của Alert:

Alert(String title)

Alert(String title, String alertText, Image alertImage, AlertType alertType)

Thành phần AlertType sử dụng âm thanh để thông báo cho người dùng biết có một sự kiện xảy ra. Ví dụ bạn có thể sử dụng AlertType để mở một đoạn âm thanh nào đó báo hiệu cho người dùng biết khi có lỗi xảy ra Thành phần AlertType bao gồm 5 loại âm thanh định sẵn là: thông báo, xác nhận, báo lỗi, thông báo và cảnh báo Ta thấy các phương thức dựng của Alert cho biết là Alert có thể bao gồm 1 tham chiếu đến một đối tượng AlertType.

Dưới đây là đoạn mã minh họa việc sử dụng Alert và AlertType

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class AlertTest extends MIDlet implements ItemStateListener,
CommandListener {
        private Display display;    // Reference to display object
        private Form fmMain; // Main form
        private Command cmExit;  // Command to exit the MIDlet
        private ChoiceGroup cgSound; // Choice group
        public AlertTest() {
                display = Display.getDisplay(this);
                // Create an exclusive (radio) choice group
                gSound = new ChoiceGroup("Choose a sound", Choice.EXCLUSIVE);
                // Append options, with no associated images
                cgSound.append("Info", null);
                gSound.append("Confirmation", null);
                gSound.append("Warning", null);
                gSound.append("Alarm", null);
                gSound.append("Error", null);
                mExit = new Command("Exit", Command.EXIT, 1);
                // Create Form, add components, listen for events
                fmMain = new Form("");
                fmMain.append(cgSound);
                fmMain.addCommand(cmExit);
                mMain.setCommandListener(this);
                fmMain.setItemStateListener(this);
        }
        public void startApp() {
                display.setCurrent(fmMain);
        }
        public void pauseApp() {}
        public void destroyApp(boolean unconditional) {}
        public void commandAction(Command c, Displayable s) {
                if (c == cmExit) {
                        destroyApp(false);
                        notifyDestroyed();
                }
        }
        public void itemStateChanged(Item item) {
                Alert al = null;
                switch (cgSound.getSelectedIndex()){
                        case 0:
```

```
                    al = new Alert("Alert sound", "Info sound",  null,
                    AlertType.INFO); break;
                case 1:
                    al = new Alert("Alert sound", "Confirm",  null,
                    AlertType.INFO);
                    break;
                case 2:
                    al = new Alert("Alert sound", "Warning",  null,
                    AlertType.INFO); break;
                case 3:
                    al = new Alert("Alert sound", "Alarm", null,
                AlertType.INFO);
                    break;
                case 4:
                    al = new Alert("Alert sound", "Error", null,
                AlertType.INFO);
                    break;
            }
            if (al != null) {
                // Wait for user to acknowledge the alert
                al.setTimeout(Alert.FOREVER);
                // Display alert, show main form when done
                display.setCurrent(al, fmMain);
            }
        }
}
```

d) Ticker

Thành phần Ticker được dùng để thể hiện một đoạn chuỗi chạy theo chiều ngang. Tham số duy nhất của thành phần Ticker là đoạn văn bản được trình bày. Tốc độ và chiều cuốn được xác định bởi việc cài đặt trên thiết bị nào.

Phương thức dựng của Ticker:

Ticker(String str);

Từ cây phân cấp các thành phần thể hiện trên thiết bị, ta thấy là thành phần Ticker không là lớp con của lớp Screen mà Ticker là một biến của lớp Screen. Điều này có

nghĩa là một Ticker có thể được gắn vào bất cứ lớp con của lớp Screen bao gồm cả Alert

Dưới đây là đoạn mã minh họa việc sử dụng một Ticker

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class TickerTest extends MIDlet implements CommandListener {
        private Display display; // Reference to Display object
        private List lsProducts; // Products
        private Ticker tkSale; // Ticker
        private Command cmExit;  // Command to exit the MIDlet
        public TickerTest() {
                display = Display.getDisplay(this);
                cmExit = new Command("Exit", Command.SCREEN, 1);
                tkSale = new Ticker("Sale: Real Imitation Cuban Cigars...10 for $10");
                lsProducts = new List("Products", Choice.IMPLICIT);
                lsProducts.append("Wicker Chair", null);
                lsProducts.append("Coffee Table", null);
                lsProducts.addCommand(cmExit);
                lsProducts.setCommandListener(this);
                lsProducts.setTicker(tkSale);
        }
        public void startApp() {
                display.setCurrent(lsProducts);
        }
        public void pauseApp() {}
        public void destroyApp(boolean unconditional) {}

        public void commandAction(Command c, Displayable s) {
                if (c == List.SELECT_COMMAND) {
                    switch (lsProducts.getSelectedIndex()) {
                        case 0:
                                System.out.println("Chair selected");
                                break;

                        case 1:
                                 System.out.println("Table selected");
                                break;
```

```
            }
        }
        else if (c == cmExit) {
            destroyApp(true);
            notifyDestroyed();
        }
    }
}
```

3. Giao diện người dùng cấp thấp

3.1 Các hàm API ở mức thấp

Mặc dù các hàm API cấp cao cung cấp một tập đầy đủ các thành phần để xây dựng giao diện ứng dụng người dùng. Tuy nhiên các thành phần cấp cao không cung cấp phương tiện để vẽ trực tiếp lên thiết bị thể hiện. Vì thiếu khả năng này nên các ứng dụng được tạo ra sẽ gặp nhiều giới hạn. Ví dụ hầu hết các nhà phát triển game di động dựa trên khả năng vẽ các đường thẳng và các hình dạng như là một phần tích hợp quá trình phát triển. Nếu các hàm API cấp cao cho phép chúng ta tạo ra giao diện cho các ứng dụng theo chuẩn, thì các hàm API cấp thấp cho phép chúng ta có thể thể hiện các ý tưởng của mình.

Canvas và Graphics là 2 lớp trái tim của các hàm API cấp thấp. Chúng ta sẽ làm tất cả các công việc bằng tay. Canvas là một khung vẽ cho phép người phát triển có khả năng vẽ lên thiết bị trình bày

cũng như là việc xử lý sự kiện. Còn lớp Graphics cung cấp các công cụ thật sự để vẽ như drawRoundRect() và drawString()

3.2 Lớp Canvas và kỹ thuật xử lý đồ họa:

Lớp Canvas cung cấp một khung vẽ cho phép tạo ra giao diện tùy biến người dùng. Một số lượng lớn các phương thức trong lớp này được dùng để xử lý sự kiện, vẽ ảnh và chuỗi lên thiết bị hiển thị. Trong phần này sẽ bao gồm các mục:

Hệ thống tọa độ

Tạo đối tượng Canvas

Vẽ lên trên đối tượng Canvas

Xử lý các sự kiện hành động

Xử lý các sự kiện phím nhấn

Xử lý sự kiện hành động của Game

Xử lý sự kiện con trỏ

Chúng ta sẽ tạo ra 2 ứng dụng MIDlet để minh họa khả năng của lớp Canvas. Ứng dụng đầu tiên là KeyMapping sẽ minh họa làm thế nào để chụp, nhận ra và xử lý mã phím nhấn và các sự kiện có liên quan đến Game. Ứng dụng còn lại là ScratchPad sẽminh họa làm thế nào để thao tác các sự kiện con trỏ để tạo ra một chương trình vẽ đường thẳng đơn giản

a) **Hệ thống trục tọa độ**

Mục tiêu đầu tiên của chúng ta là làm quen với hệ thống trục tọa độđể làm việc với thiết bị thể hiện. Hệ thống tọa độ cho lớp Canvas có tâm tọa độ là điểm trái trên của thiết bị trình bày. Giá trị x tăng dần về phía phải, giá trị y tăng dần khi đi xuống phía dưới. Khi vẽđộ dày bút vẽ là một điểm ảnh

Các phương thức sau đây sẽ giúp xác định chiều rộng và chiều cao của canvas:

int getWidth(): xác định chiều rộng của canvas

int getHeight (): xác định chiều cao của canvas

Chiều rộng và chiều cao của Canvas cũng đại diện cho toàn bộ diện tích khung vẽ có thể trên thiết bị trình bày. Nói cách khác, chúng không thể chỉ định kích thước cho canvas, mà phần mềm trên một thiết bị MIDP sẽ trả về diện tích lớn nhất có thể có đối với một thiết bị cho trước

b) Tạo một đối tượng Canvas

Bước đầu tiên để làm việc với một lớp Canvas là tạo ra một lớp thừa kế từ lớp Canvas

```
class TestCanvas extends Canvas implements CommandListener {
        private Command cmdExit; ...
        display = Display.getDisplay(this);
        cmdExit = new Command("Exit", Command.EXIT, 1);
        addCommand(cmdExit);
        setCommandListener(this); ...
        protected void paint(Graphics g) {
            // Draw onto the canvas
            ...
        }
}
TestCanvas canvas = new TestCanvas(this);
```

c) **Vẽ trên đối tượng Canvas**

Phương thức paint của lớp Canvas cho phép bạn vẽ các hình dạng, vẽ ảnh, xuất chuỗi. Đoạn mã sau minh họa việc xóa màn hình thể hiện bằng một màu trắng

```
protected void paint(Graphics g) {
        // Set background color to white
        g.setColor(255, 255, 255);
        // Fill the entire canvas
        g.fillRect(0, 0, getWidth(), getHeight());
```

}

Chúng ta có thể sử dụng một tham chiếu đến một đối tượng Graphics bên trong thân phương thức paint() để thực hiện công việc vẽ thực sự

d**) Sự kiện hành động**

Cũng như các thành phần Form, List, và TextBox, một Canvas có thể xử lý các sự kiện Command. Chúng ta có thể xử lý các sự kiện Command trên thành phần Canvas cung cách như các thành phần khác

Đoạn mã sau minh họa việc xử lý sự kiện Command trên thành phần Canvas

```
class TestCanvas extends Canvas implements CommandListener {
    private Command cmdExit;
    ...
    display = Display.getDisplay(this);
    cmdExit = new Command("Exit", Command.EXIT, 1);
    addCommand(cmdExit);
    setCommandListener(this); ...
    protected void paint(Graphics g) {
        // Draw onto the canvas
        ...
    }
    public void commandAction(Command c, Displayable d) {
        if (c == cmdExit)
        ...
    }
```

}

e) Mã phím

Trong trường hợp xử lý các hành động của các phím mềm, một Canvas có thể truy cập đến 12 mã phím. Những mã này được đảm bảo luôn luôn có trên bất kỳ các thiết bị MIDP nào

KEY_NUM0

KEY_NUM1

KEY_NUM2

KEY_NUM3

KEY_NUM4

KEY_NUM5

KEY_NUM6

KEY_NUM7

KEY_NUM8

KEY_NUM9

KEY_STAR

KEY_POUND

Năm phương thức để xử lý các mã phím là:

void keyPressed(int keyCode)

void keyReleased(int keyCode)

void keyRepeated(int keyCode)

boolean hasRepeatEvents()

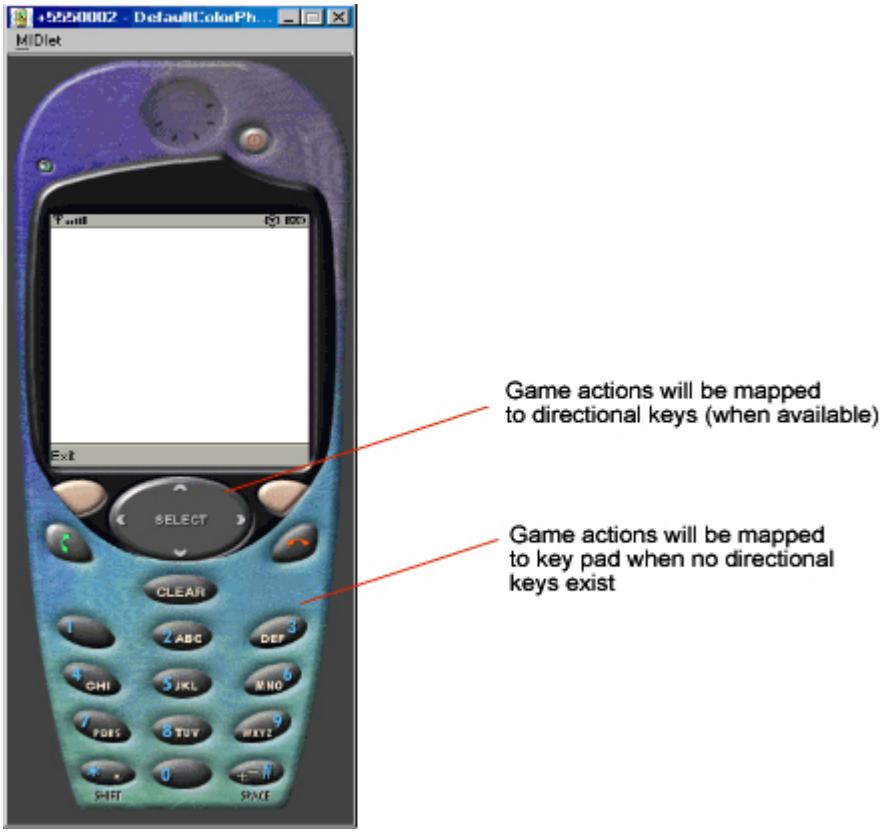
String getKeyName(int keyCode)

f**) Các hành động trong xử lý các trò chơi**

MIDP thường được sử dụng để tạo các trò chơi trên nền Java. Các hằng số sau đã được định nghĩa để xử lý các sự kiện có liên quan đến trò chơi trong MIDP

UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, GAME_D

Nói một cách đơn giản thì các giá trị này được ánh xạ thành các phím mũi tên chỉhướng của thiết bị, nhưng không phải tất cả các thiết bị di động đều có những giá trịnày. Nếu một thiết bị di động thiếu các phím mũi tên thì các hành động của trò chơi sẽđược ánh xạ vào các nút bấm, ví dụ phím trái được ánh xạ vào phím số 2, phím phải được ánh xạ vào phím số 5, và cứ tiếp tục như thế. Hình dưới đây cho thấy các hành động của trò chơi sẽ được ánh xạ lên một thiết bị di động dựa trên khả năng của các phím chỉ hướng



g) Xác định các hành động của trò chơi

Đoạn mã sau đây mô tả một cách xác định các hành động của trò chơi để từ đó gọi các phương thức thích hợp dựa trên các hành động xảy ra

```
protected void keyPressed(int keyCode) {
        switch (getGameAction(keyCode)) {
            case Canvas.FIRE:
                    shoot();
                    break;
            case Canvas.RIGHT:
                        goRight();
                        break;
        ... }
}
```

Một lựa chọn nữa là có thể tạo một tham chiếu cho mỗi hành động của trò chơi thông qua quá trình khởi tạo giá trị cho các biến

```
// Initialization
keyFire = getKeyCode(FIRE);
keyRight = getKeyCode(RIGHT);
keyLeft = getKeyCode(LEFT); ...
// Runtime
protected void keyPressed(int keyCode) {
    if (keyCode == keyFire) shoot();
    else if (keyCode == keyRight) goRight()
    ...
}
```

Đoạn mã dưới đây minh họa một số chức năng của Canvas và cách xử lý phím

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class KeyMapping extends MIDlet {
        private Display display; // The display
        private KeyCodeCanvas canvas;  // Canvas
```

```
    public KeyMapping() {
        display = Display.getDisplay(this);
        canvas = new KeyCodeCanvas(this);
    }
    protected void startApp() {
        display.setCurrent( canvas );
    }
    protected void pauseApp() {}
    protected void destroyApp( boolean unconditional ) {}
    public void exitMIDlet() {
        destroyApp(true);
        notifyDestroyed();
    }
}

/*------------------------------* Class KeyCodeCanvas*------------------------------*/
class KeyCodeCanvas extends Canvas implements CommandListener {
    private Command cmExit; // Exit midlet
    private String keyText = null; // Key code text
    private KeyCodes midlet;
        public KeyCodeCanvas(KeyCodes midlet) {
            this.midlet = midlet;
                // Create exit command and listen for events
            cmExit = new Command("Exit", Command.EXIT, 1);
            addCommand(cmExit);
            setCommandListener(this);
    }

    protected void paint(Graphics g) {
          // Clear the background (to white)
        g.setColor(255, 255, 255);
       g.fillRect(0, 0, getWidth(), getHeight());
       // Set color and draw text
        if (keyText != null) {
       // Draw with black pen
        g.setColor(0, 0, 0);
        // Center text
        g.drawString(keyText, getWidth()/2, getHeight()/2, Graphics.TOP |
Graphics.HCENTER);
    }
    }
    public void commandAction(Command c, Displayable d) {
```

```
                if (c == cmExit)  midlet.exitMIDlet();
        }
        protected void keyPressed(int keyCode) {
                keyText = getKeyName(keyCode);
                repaint();
        }
}
```

h) Sự kiện con trỏ

Trong phần này chúng ta sẽ quản lý sự kiện con trỏ trong một Canvas. Những sựkiện này được thiết kếđể làm thuận tiện cho việc tương tác với các thiết bị có dạng con trỏ. Một số phương thức được cung cấp nhằm hỗ trợ cho việc xử lý sự kiện con trỏ:

boolean hasPointerEvents()

boolean hasPointerMotionEvents()

void pointerPressed(int x, int y)

void pointerReleased(int x, int y)

void pointerDragged(int x, int y)

Các phương thức trên có thể tự giải thích chức năng thông qua tên của chính mình. Ví dụ như phương thức hasPointerMotionEvents() trả về một giá trị có kiểu boolean nhằm chỉ rõ rằng thiết bị di động có hỗ trợ khái niệm "nhấp chuột và rê" hay không

Đoạn chương trình dưới đây minh họa việc sử dụng các sự kiện con trỏ để thực hiện một chương trình vẽ đơn giản

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ScratchPad extends MIDlet {
        private Display display; // Display object
        private ScratchPadCanvas canvas; // Canvas
        public ScratchPad() {
                display = Display.getDisplay(this);
                canvas = new ScratchPadCanvas(this);
        }
        protected void startApp() {
```

```
            display.setCurrent( canvas );
        }
        protected void pauseApp() {}
        protected void destroyApp( boolean unconditional ){}

        public void exitMIDlet() {
            destroyApp(true);
            notifyDestroyed();
        }
}

/*-----------* Class ScratchPadCanvas * * Pointer event handling *---------------*/
class ScratchPadCanvas extends Canvas implements CommandListener {
        private Command cmExit;  // Exit midlet
        private Command cmClear;  // Clear display
        private int startx = 0;
        // Where pointer was clicked
        starty = 0;  currentx = 0;
        // Current location
        currenty = 0;
        private ScratchPad midlet;
        private boolean clearDisplay = true;
        public ScratchPadCanvas(ScratchPad midlet) {
                this.midlet = midlet;
                // Create exit command and listen for events
                cmExit = new Command("Exit", Command.EXIT, 1);
                cmClear = new Command("Clear", Command.SCREEN, 1);
                addCommand(cmExit);
                addCommand(cmClear);
                setCommandListener(this);
        }
        protected void paint(Graphics g) {
                // Clear the background (to white)
                if (clearDisplay) {
                    g.setColor(255, 255, 255);
                    g.fillRect(0, 0, getWidth(), getHeight());
                    clearDisplay = false;
                    startx = currentx = starty = currenty = 0;
                    return;
                }
                // Draw with black pen
                g.setColor(0, 0, 0);
```

```
            // Draw line
            g.drawLine(startx, starty, currentx, currenty);
            // New starting point is the current position
            startx = currentx;
            starty = currenty;
        }

        public void commandAction(Command c, Displayable d) {
            if (c == cmExit) midlet.exitMIDlet();
            else if (c == cmClear) {
                clearDisplay = true;
                repaint();
            }
        }
        protected void pointerPressed(int x, int y) {
            startx = x;
            starty = y;
        }

        protected void pointerDragged(int x, int y) {
            currentx = x;
            currenty = y;
            repaint();
         }
}
```

3.3 Lớp Graphics

Chúng ta sử dụng đối tượng Graphics để vẽ lên một Canvas.

a) Hỗ trợ màu

Một ứng dụng MIDP chỉ có một đối tượng Display. Đối tượng này được dùng để lấy thông tin của màn hình hiển thị hiện tại, ví dụ như số màu hỗ trợ và các phương thức để yêu cầu các đối tượng được hiển thị. Đối tượng Display đơn giản là một bộ quản lý sự hiển thị của thiết bị và điều khiển những gì sẽđược hiển thị ra trên thiết bị.

Có hai phương thức chúng ta cần quan tâm đến:

boolean isColor()

int numColors()

Phương thức đầu tiên cho biết thiết bị có hỗ trợ hiển thị màu hay không. Nếu có thì phương thức thứ 2 sẽ được gọi để xác định số màu được hỗ trợ. Các phương thức tiếp theo dưới đây để lấy về màu và thiết lập màu ưa thích của bạn.

void setColor(int RGB)

void setColor(int red, int green, int blue)

int getColor()

int getBlueComponent()

int getGreenComponent()

int getRedComponent()

void setGrayScale(int value)

int getGrayScale()

Chú ý ta có thể xác định màu bằng 2 cách.

Cách 1: Xác định một số nguyên đại diện cho 3 giá trị của màu là đỏ, xanh lá cây và xanh dương với 8 bit cho mỗi màu.

Cách 2: Dùng từng tham số riêng biệt để xác định mỗi màu. Khi sử dụng một giá trịđể lưu giữ màu, thì màu đỏ sẽ chiếm 8 bit đầu kể từ bên trái, tiếp theo là 8 bit dành cho màu xanh lá cây, sau cùng là màu xanh dương.

Dưới đây là cách thiết lập màu chỉ sử dụng một số nguyên:

```
int red = 0,    green = 128, blue = 255;
...
g.setColor((red << 16) | (green << 8) | blue);
```

Và ta có thể xác định màu bằng cách thiết lập giá trị cho 3 tham số:

```
g.setColor(red, green, blue);
```

b) Loại nét vẽ

Bạn có thể chọn nét khi vẽđường thẳng, cung và hình chữ nhật trên thiết bị hiển thị. Dưới đây là các phương thức dùng để thiết lập loại nét vẽ

int getStrokeStyle()

void setStrokeStyle(int style)

Hai kiểu nét vẽ được định nghĩa trong lớp Graphics là nét chấm, và nét liền

g.setStrokeStyle(Graphics.DOTTED

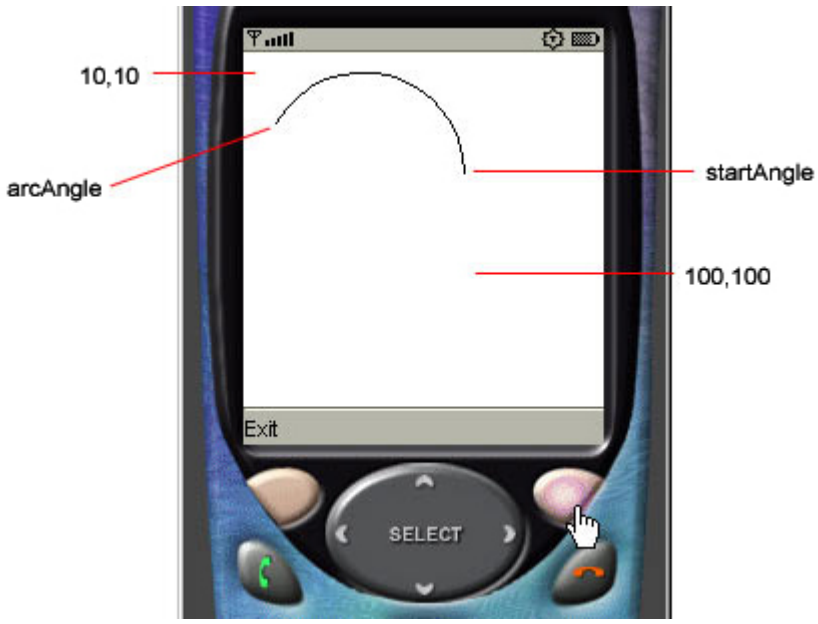
g.setStrokeStyle(Graphics.SOLID);

c) Vẽ cung

Khi vẽ một cung, bạn có thể vẽ nó chỉ có đường bao xung quanh hay yêu cầu nó được tô bên trong. Bạn có thể bắt đầu bằng cách chỉđịnh chiều bao quanh bên ngoài của một hình hộp chữ nhật tưởng tượng. Góc bắt đầu xác định vị trí bắt đầu vẽkhung, với giá trị 0 được xác định tại thời điểm 3 giờ. Giá trị dương tính theo ngược chiều kim đồng hồ. Góc của cung chỉ ra rằng có bao nhiêu độ được vẽ tính từ góc ban đầu, đi theo ngược chiều kim đồng hồ. Để hiểu rõ những phần này chúng ta hãy cùng xem1 ví dụ sau:

g.drawArc(10, 10, 100, 100, 0, 150);

Đoạn mã trên yêu cầu vẽ một cung, cung này được bao bởi một hình chữ nhật có tọa độđiểm trái trên là (10, 10), chiều rộng và chiều dài là 100, góc bắt đầu là 0, góc kết thúc là 150



Một số các phương thức dùng để vẽ cung

void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)

Dưới đây là đoạn mã minh họa viêc sử dụng các hàm trên để vẽ một cung

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DrawShapes extends MIDlet {
        private Display display; // The display
        private ShapesCanvas canvas; // Canvas
        public DrawShapes() {
                display = Display.getDisplay(this);
                canvas = new ShapesCanvas(this);
        }
        protected void startApp() {
                display.setCurrent( canvas );
        }
        protected void pauseApp() {}
        protected void destroyApp( boolean unconditional ) {}

        public void exitMIDlet() {
            destroyApp(true);
```

```
            notifyDestroyed();
        }
}

class ShapesCanvas extends Canvas implements CommandListener {
        private Command cmExit;  // Exit midlet
        private DrawShapes midlet;
        public ShapesCanvas(DrawShapes midlet) {
                this.midlet = midlet;
                // Create exit command and listen for events
                cmExit = new Command("Exit", Command.EXIT, 1);
                addCommand(cmExit);
                setCommandListener(this);
        }
        protected void paint(Graphics g) {
            // Clear background to white
                g.setColor(255, 255, 255);
                g.fillRect(0, 0, getWidth(), getHeight());
                // Black pen
                g.setColor(0, 0, 0);
                // Start at 3 o'clock and rotate 15 degrees
                g.drawArc(10, 10, 100, 100, 0, 150);
            // Fill the arc
                // g.fillArc(10, 10, 100, 100, 0, 150);
                // Start at 12 o'clock and rotate 150 degrees
                // g.drawArc(10, 10, 100, 100, 90, 150);
                // Change the size of the bounding box
                // Start at 12 o'clock and rotate 150 degrees
                // g.drawArc(15, 45, 30, 70, 90, 150); }
        public void commandAction(Command c, Displayable d) {
            if (c == cmExit) midlet.exitMIDlet();
        }
}
```

d) Vẽ hình chữ nhật

Cũng giống như cung thì hình chữ nhật có thể chỉđược vẽ viền bao quanh hoặc tô bên trong. Bên cạnh đó bạn có thể vẽ hình chữ nhật đó có 4 góc là tròn hoặc là vuông. Dưới đây là một số phương thức để vẽ hình chữ nhật:

void drawRect(int x, int y, int width, int height)

void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

void fillRect(int x, int y, int width, int height)

void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)

Khi vẽ hình chữ nhật có 4 góc là tròn thì bạn phải xác định đường kính theo chiều ngang (arcWidth) và đường kính theo chiều dọc (arcHeight). Những tham số này được định nghĩa độ sắc nét của cung theo mỗi chiều. Giá trị càng lớn thể hiện một cung tăng dần, ngược lại là một đường cong hẹp

e) Font chữ

Phần sau đây cũng quan trọng không kém là cách sử dụng font chữđược hỗ trợ bởi giao diện cấp thấp của ứng dụng MIDP. Sau đây là một số các phương thức dựng của lớp Font

Font getFont(int face, int style, int size)
Font getFont(int fontSpecifier)
Font getDefaultFont()

Một số thuộc tính của lớp Font

FACE_SYSTEM
FACE_MONOSPACE
FACE_PROPORTIONAL
STYLE_PLAIN
STYLE_BOLD
STYLE_ITALIC
STYLE_UNDERLINED
SIZE_SMALL
SIZE_MEDIUM
SIZE_LARGE

Các tham số kiểu dáng có thể được kết hợp thông qua toán tử | . Ví dụ

Font font = Font.getFont(Font.FACE_SYSTEM ,Font.STYLE_BOLD |

Font.STYLE_ITALIC, Font.SIZE_SMALL);

Sau khi bạn có một tham chiếu đến một đối tượng Font, bạn có thể truy vấn nó đểxác định thông tin của các thuộc tính của nó.

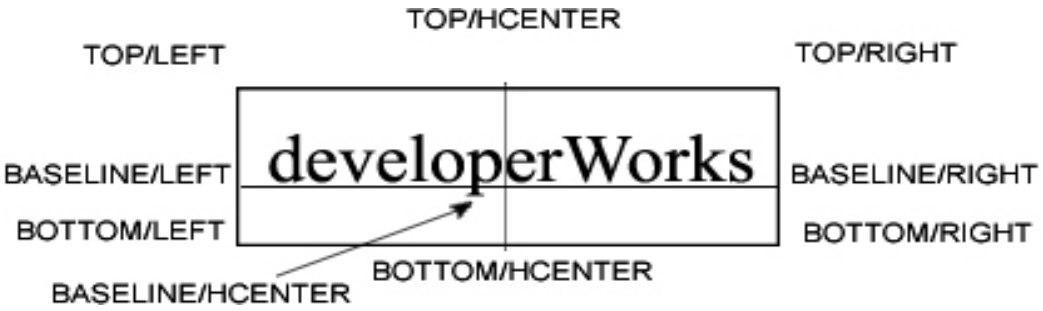
int getFace()
int getStyle()
int getSize()
boolean isPlain()
boolean isBold()
boolean isItalic()
boolean isUnderlined()

Kích thước của các font chữ được xác định bởi chiều cao của font chữ, bề dài tính bằng điểm ảnh của một chuỗi ký tự trong một font xác định. Một số các phương thức sau hỗ trợ khi tương tác với một đối tượng font

int getHeight()
int getBaselinePosition()
int charWidth(char ch)
int charsWidth(char[] ch, int offset, int length)
int stringWidth(String str)
int substringWidth(String str, int offset, int length)

f) Điểm neo

Để xác định tọa độ x, y của chuỗi ký tựđược hiển thị, thì điểm neo cho phép chúng ta chỉ ra vị trí muốn đặt tọa độ (x,y) trên hình chữ nhật bao quang chuối ký tự



Có 6 điểm neo được định nghĩa trước, 3 theo chiều dọc và 3 theo chiều thẳng đứng. Khi xác định điểm neo để vẽ chuỗi (các điểm neo thường được sử dụng thành từng cặp), ta phải chọn một điểm hoành độ và một điểm tung độ. Các điểm neo được định nghĩa như ở dưới đây

Chiều ngang

LEFT (Bên trái)

HCENTER (Chính giữa của chiều ngang)

RIGHT (Bên phải)

Chiều dọc

TOP (Ở trên)

BASELINE (Đường thẳng cơ sở)

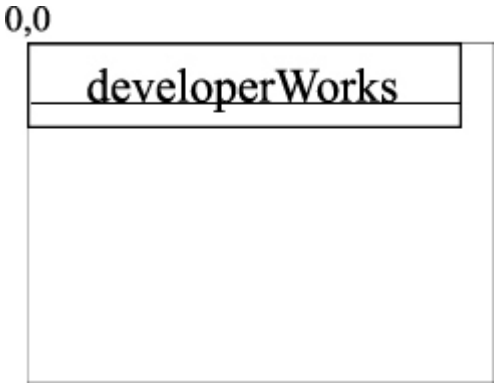
BOTTOM (Ở dưới)

Khi sử dụng điểm neo thì cần phải chỉ ra tọa độ x, y của hình chữ nhật bao quanh.

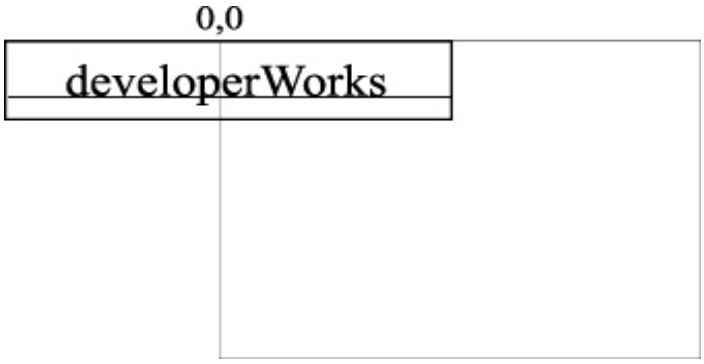
Ví dụ

g.drawString("developerWorks", 0, 0 , Graphics.TOP | Graphics.LEFT);

Hình dưới đây mô tả kết quả của hàm trên



Bằng cách thay đổi điểm neo, chúng ta có thể thay đổi vị trí hiển thị của chuỗi ký tự trên thiết bị di động. Ví dụ tiếp theo chúng ta sẽ minh họa tiếp khi thay đổi điểm neo thì vị trí của chuỗi ký tự cũng thay đổi theo:



g) Vẽ các chuỗi ký tự

Sau khi tìm hiểu về font và các điểm neo, bạn đã có thể vẽ chuỗi ký tự ra màn hình thông qua một số các phương thức sau:

void drawChar(char character, int x, int y, int anchor)
void drawChars(char[] data, int offset, int length, int x, int y, int anchor)
void drawString(String str, int x, int y, int anchor)
void drawSubstring(String str, int offset, int len, int x, int y, int anchor)

Ví dụ:

```
protected void paint(Graphics g) {
        // Get center of display
        int xcenter = getWidth() / 2,
        ycenter = getHeight() / 2;
         // Choose a font
        g.setFont(Font.getFont(Font.FACE_SYSTEM, Font.STYLE_BOLD,
        Font.SIZE_SMALL));
    // Specify the center of the text (bounding box) using the anchor point
    g.drawString("developerWorks", xcenter, ycenter, Graphics.BASELINE |
Graphics.HCENTER);
}
```

Tiếp theo là ví dụ minh họa việc sử dung font và xuất chuỗi ra thiết bị hiển thị

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class FontViewer extends MIDlet {
        protected Display  display; // The display
        protected PrefsForm fmPrefs;  // Form to choose font prefs
        protected FontCanvas cvFont;  // Canvas to display text (in preferred font)
        public FontViewer() {
            display = Display.getDisplay(this);
            cvFont = new FontCanvas(this);
            fmPrefs = new PrefsForm("Preferences", this);
        }
        protected void startApp() {
            showCanvas();
        }

        protected void showCanvas() {
            display.setCurrent(cvFont);
        }
        protected void pauseApp() {}
```

```
        protected void destroyApp( boolean unconditional ) {}

        public void exitMIDlet() {
            destroyApp(true);
            notifyDest
        }
}

/*---------------------------------------
* FontCanvas.java
* ---------------------------------------*/
import javax.microedition.lcdui.*;
class FontCanvas extends Canvas implements CommandListener {
        private int         face, // Font face
                        style, // style
                        size; // size
        private String text = "developerWorks"; // Text to display in preferred font
        private Command cmExit;  // Exit midlet
        private Command cmPrefs;  // Call the preferences form
        private FontViewer midlet; // Reference to the main midlet
        public FontCanvas(FontViewer midlet) {
                this.midlet = midlet;
                // Create commands and listen for events
                cmExit = new Command("Exit", Command.EXIT, 1);
                cmPrefs = new Command("Prefs", Command.SCREEN, 2);
                addCommand(cmExit);
                addCommand(cmPrefs);
                setCommandListener(this);
        }
        protected void paint(Graphics g) {
            // Clear the display
                g.setColor(255, 255, 255);
                // White pen
                g.fillRect(0, 0, getWidth(), getHeight());
                g.setColor(0, 0, 0);           // Black pen
            // Use the user selected font preferences
                g.setFont(Font.getFont(face, style, size));
                // Draw text at center of display
                g.drawString(text, getWidth()/2, getHeight()/2,Graphics.BASELINE |
        Graphics.HCENTER);
        }
        protected void setFace(int face) {
```

```
            this.face = face;
        }
        protected void setStyle(int style) {
            this.style = style;
        }
        protected void setSize(int size) {
           this.size = size;
        }
        public void setText(String text) {
            this.text = text;
        }
        public int getFace() {
            return face;
        }
        public int getStyle() {
            return style;
        }
        public int getSize() {
            return size;
        }
        public void commandAction(Command c, Displayable d) {
            if (c == cmExit) midlet.exitMIDlet();
            else if (c == cmPrefs) midlet.display.setCurrent(midlet.fmPrefs);
        }
}
```

h) Vẽ ảnh

Lớp Graphics cung cấp 1 phương thức dùng để vẽ ảnh:

drawImage(Image img, int x, int y, int anchor)

Chúng ta cũng áp dụng từng bước khi vẽ ảnh cũng giống như khi xuất chuỗi ra màn hình. Đối với cả 2 thì chúng ta đều phải bắt đầu bằng việc thiết lập tọa độ x, y cũng nhưđiểm neo. Danh sách các điểm neo cho việc hiển thị ảnh cũng không khác mấy so với việc xuất chuỗi, tuy nhiên không giống với việc xuất chuỗi thì một bức ảnh có một điểm trung tâm. Ví thếVCENTER được thay thế cho gia trị BASELINE khi làm việc với ảnh

Chiều ngang

LEFT (Bên trái)

HCENTER (Điểm chính giữa theo chiều ngang)

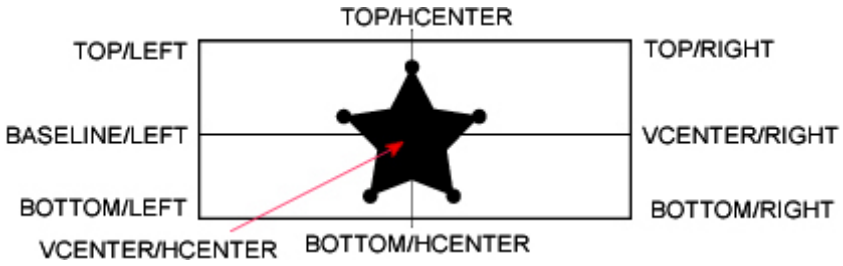
RIGHT (Bên phải)

Chiều dọc

TOP (Điểm trên)

VCENTER (Điểm chính giữa theo chiều dọc)

BOTTOM (Bên dưới



Trong các phần trước, chúng ta đã tạo ra các ứng dụng MIDP cho việc trình bày một tấm ảnh đọc từ một nguồn tài nguyên là một tập tin. Loại ảnh này không cho phép thay đổi, và vì vậy còn được biết với tên là "ảnh không thể thay đổi". Đối với ví dụ sau đây, chúng ta sẽ tạo ra một tấm ảnh bằng cách cấp phát bộ nhớ cho tấm ảnh, để lấy tham chiếu đến một đối tượng Graphics, và chúng ta sẽ tự vẽ nội dung tấm ảnh. Loại ảnh này còn được biết với một cái tên là "ảnh có thể biến thay đổi được"

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DrawImage extends MIDlet {
        private Display display; // The display
        private ImageCanvas canvas;  // Canvas
        public DrawImage() {
                display = Display.getDisplay(this);
                canvas = new ImageCanvas(this);
        }
        protected void startApp() {
```

```
            display.setCurrent( canvas );
        }
        protected void pauseApp(){}
        protected void destroyApp( boolean unconditional ) {}

        public void exitMIDlet() {
            destroyApp(true);
            notifyDestroyed();
        }
}
/*---------------------------------
* Class ImageCanvas *
* ---------------------------------*/
class ImageCanvas extends Canvas implements CommandListener {

        private Command cmExit;  // Exit midlet
        private DrawImage midlet;
        private Image im = null;
        private String message = "developerWorks";
        public ImageCanvas(DrawImage midlet) {
                this.midlet = midlet;
            // Create exit command and listen for events
                cmExit = new Command("Exit", Command.EXIT, 1);
                addCommand(cmExit);
                setCommandListener(this);
                try{
                    // Create mutable image
                    im = Image.createImage(100, 20);
                    // Get graphics object to draw onto the image
                    Graphics graphics = im.getGraphics();
                    // Specify a font face, style and size

                        Font font = Font.getFont(Font.FACE_SYSTEM,
                        Font.STYLE_PLAIN, Font.SIZE_MEDIUM);
                        graphics.setFont(font);
                         // Draw a filled (blue) rectangle, with rounded corners
                        graphics.setColor(0, 0, 255);
                         graphics.fillRoundRect(0,0, im.getWidth()-1, im.getHeight()-1,
                        20, 20);
                        // Center text horizontally in the image. Draw text in white
                        graphics.setColor(255, 255, 255);
                        graphics.drawString(message,   (im.getWidth() / 2) -
```

```
                font.stringWidth(message) / 2), (im.getHeight() / 2) -
                (font.getHeight() / 2),  Graphics.TOP | Graphics.LEFT);
            }
            catch (Exception e) {
                System.err.println("Error during image creation");
            }
        }

        protected void paint(Graphics g) {
            // Clear the display
            g.setColor(255, 255, 255);
            g.fillRect(0, 0, getWidth(), getHeight());
            // Center the image on the display
            if (im != null)
            g.drawImage(im, getWidth()/2, getHeight()/2,Graphics.VCENTER |
            Graphics.HCENTER);
        }
        public void commandAction(Command c, Displayable d) {
            if (c == cmExit) midlet.exitMIDlet();
        }
}
```

i) Một số các phương thức khác của lớp Graphics:

clip() và translate() là 2 phương thức của lớp Graphics. Một vùng hiển thị được cắt xén được định nghĩa là khu vực hiển thị của thiết bị di động, vùng này sẽ được cập nhật trong suốt thao tác vẽ lại. Dưới đây là một số phương thức hỗ trợ cho việc xén một vùng hiển thị

```
void setClip(int x, int y, int width, int height)
void clipRect(int x, int y, int width, int height)
int getClipX()
int getClipY()
int getClipWidth()
int getClipHeight()
```

translate() là một phương thức được sử dụng có liên quan đến hệ thống trục tọa độ. Chúng ta có thể tịnh tiến hệ trục tọa độ đến một điểm x, y khác. Một số phương thức hỗ trợ cho việc tịnh tiến hệ trục tọa độ

```
void translate(int x, int y)
int getTranslateX()
```

int getTranslateY()

Chương III: HỆ THỐNG QUẢN LÝ BẢN GHI

(Record Management System - RMS)

MIDP không sử dụng hệ thống file để lưu trữ dữ liệu. Thay vào đó MIDP lưu toàn bộ thông tin vào non-volatile memory bằng hệ thống lưu trữ gọi là Record Management System (RMS).

1. Lưu trữ cố định thông qua Record Store

RMS là hệ thống được tổ chức và quản lý dưới dạng các record (bản ghi). Mỗi bản ghi (sau này gọi là Record) có thể chứa bất kỳ loại dữ liệu nào, chúng có thể là kiểu số nguyên, chuỗi ký tự hay có thể là một ảnh và kết quả là một Record là một chuỗi (mảng) các byte. Nếu bạn mã hoá dữ liệu của bạn dưới dạng nhị phân (binary), bạn có thể lưu trữ dữ liệu bằng Record sau đó đọc dữ liệu từ Record và khôi phục lại dữ liệu ban đầu. Tất nhiên kích thước dữ liệu của bạn không được vượt quá giới hạn qui định của thiết bị di động. RMS lưu dữ liệu gần như một cơ sở dữ liệu, bao gồm nhiều dòng, mỗi dòng lại có một số định danh duy nhất

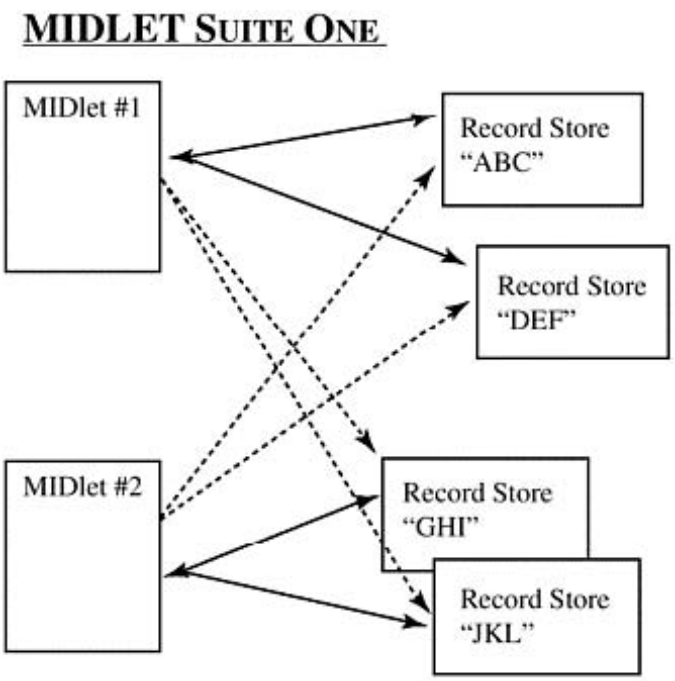
Một cơ sở dữ liệu kiểu bản ghi

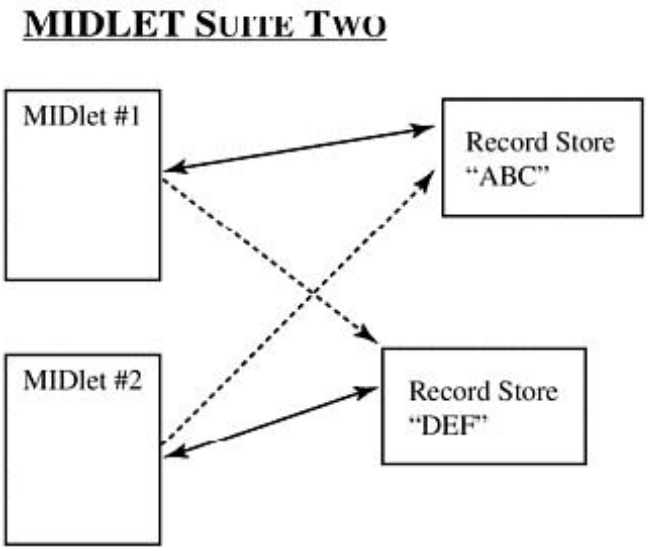
Record ID	Data
1	Array of bytes
2	Array of bytes
3	Array of bytes
…	

Một tập các bản ghi (sau này gọi là RecordStore) là tập hợp các Record được sắp xếp có thứ tự. Mỗi Record không thểđứng độc lập mà nó phải thuộc vào một RecordStore nào đó, các thao tác trên Record phải thông qua RecordStore chứa nó. Khi tạo ra một Record trong RecordStore, Record được gán một sốđịnh danh kiểu số nguyên gọi là Record ID. Record đầu tiên được tạo ra sẽđược gán Record ID là 1 và sẽ

tăng thêm 1 cho các Record tiếp theo. Cần chú rằng Record ID không phải là chỉ mục (index), các thao tác xóa Record trong RecordStore sẽ không gây nên việc tính toán lại các Record ID của các Record hiện có cũng như không làm thay đổi Record ID của các Record được tạo mới, ví dụ: khi ta xóa record id 3 khi thêm một record mới sẽ có id là 4. Data là một dãy các byte đại diện cho dữ liệu cần lưu.

Tên được dùng để phân biệt các RecordStore trong bộ các MIDlet (MIDlet suite). Cần chú ý khái niệm MIDlet suite là tập các MIDlet có chung không gian tên (name space), có thể chia sẽ cùng tài nguyên (như RecordStore), các biến tĩnh (static variable) trong các lớp và các MIDlet này sẽđược đóng gói trong cùng một file .jar khi triển khai. Nếu ứng dụng của bạn chỉ có một MIDlet thì các RecordStore được sử dụng cũng phân biệt lẫn nhau bằng các tên. Tên của RecordStore có thể dài đến 32 ký tự Unicode và là duy nhất trong một MIDlet suite.





Đường liền thể hiện việc truy xuất Record store do MIDlet đó tạo ra, đường nét đứt là Record store do MIDlet khác tạo ra. Trong MIDLET Suite One, MIDlet #1 và MIDlet #2 cùng có thể truy xuất 4 Record store. MIDLET Suite One không thể truy xuất Record store trong Suite Two. Trong MIDlet Suite One tên của các Record store là duy nhấy, tuy nhiên Record store trong các MIDlet Suite khác nhau có thể dùng chung một tên.

Record Store còn có 2 thuộc tính là Version Number và Date/time Stamp, các giá trị này thay đổi khi thực hiện thêm, thay thế hay xóa một record, ngoài ra còn có

thể dùng cơ chế event handler (Listener) để phát hiện mỗi khi Record store bị thay đổi. Version number là một số integer, để biết giá trị khởi đầu cần gọi hàm getVersion() sau khi tạo một Record store. Date/time Stamp là số long integer, là số miliseconds kể từ ngày 1 tháng 1 năm 1970, chúng ta có thể biết được giá trị này thông qua hàm getLastModified().

2. Các Vấn Đề Liên Quan Đến RMS

a) Hạn chế về khả năng lưu trữ của thiết bị di động

Dung lượng vùng nhớ (non-volatile memory) dành riêng cho việc lưu trữ dữ liệu trong RMS thay đổi tùy theo thiết bị di động. Đặc tả MIDP yêu cầu rằng các nhà sản xuất thiết bị di động phải dành ra vùng nhớ có kích thước ít nhất 8K cho việc lưu trữ dữ liệu trong RMS. Đặc tả không nêu giới hạn trên cho mỗi Record. RMS cung cấp các API để xác định kích thước của mỗi Record, tổng dung lượng của RecordStore và kích thước còn lại của vùng nhớ này. Do đó trong quá trình phát triển các ứng dụng J2ME lập trình viên phải cân nhắc trong việc sử dụng vùng nhớ này.

b) Tốc độ truy xuất dữ liệu

Các thao tác trên vùng nhớ này (non-volatile memory) tất nhiên sẽ chậm hơn nhiều khi truy xuất dữ liệu trên bộ nhớ RAM (volatile memory). Nó sẽ giống như tốc độ đọc ổ cứng và tốc độ đọc từ RAM của máy tính. Vì vậy trong kỹ thuật lập trình phải thường xuyên cache dữ liệu và các thao tác liên quan đến RMS chỉ thực hiện tập trung một lần (lúc khởi động hay đóng ứng dụng).

c) Cơ chế luồng an toàn

Nếu RecordStore của chỉ được sử dụng bởi một MIDlet thì không phải lo lắng về vấn đề này vì RMS sẽ dành riêng một Thread để thực hiện các thao tác trên RecordStore. Tuy nhiên nếu có nhiều MIDlet và Thread cùng chia sẻ một RecordStore thì phải chú ý đến kỹ thuật lập trình Thread để đảm bảo không có sự xung đột dữ liệu.

3. Các Hàm API Trong RMS

RecordStore không có hàm khởi tạo.

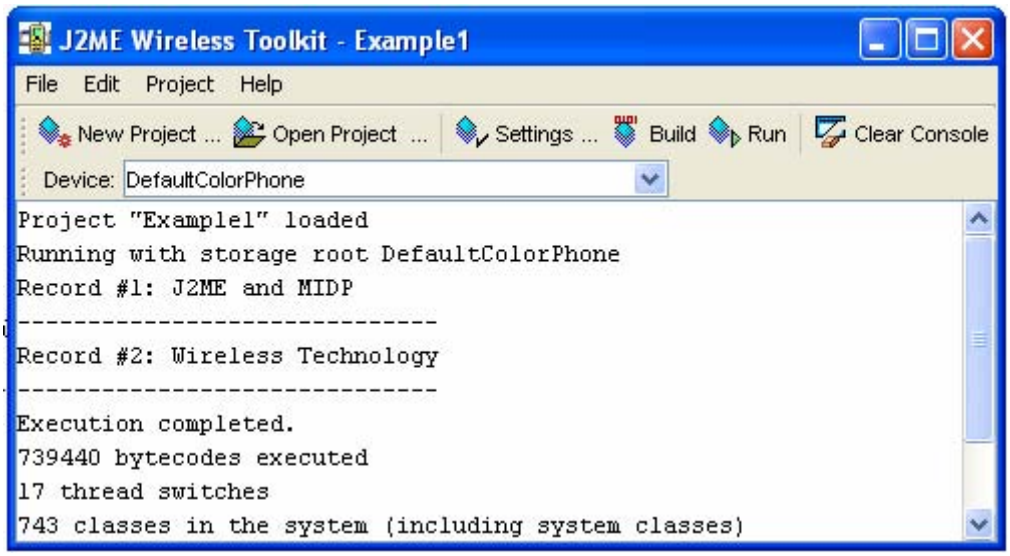
RecordStore Class: javax.microedition.rms.RecordStore	
Method	**Description**
static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary)	Mở một Recordstore, có tham số tạo Record store nếu nó chưa tồn tại.
void closeRecordStore()	Đóng RecordStore.
static void deleteRecordStore(String recordStoreName)	Xóa RecordStore.
static String[] listRecordStores()	Danh sách các RecordStore trong MIDlet suite.
int addRecord(byte[] data, int offset, int numBytes)	Thêm một record vào RecordStore.
void setRecord(int recordId, byte[] newData, int offset, int numBytes)	Đặt hoặc thay thế một record trong RecordStore.
void deleteRecord(int recordId)	Xóa một record trong RecordStore.
byte[] getRecord(int recordId)	Lấy dãy byte chứa record.
int getRecord(int recordId, byte[] buffer, int offset)	Lấy nội dung của record vào dãy byte.
int getRecordSize(int recordId)	Kích thước record.
int getNextRecordID()	Lấy record id của record mới .
int getNumRecords()	Số lượng các record.
long getLastModified()	Thời gian thay đổi gần nhất.
int getVersion()	Version của RecordStore.
String getName()	Tên của RecordStore.
int getSize()	Kích thước của RecordStore.
int getSizeAvailable()	Số byte trống cho RecordStore.
RecordEnumeration enumerateRecords(RecordFilter filter, RecordComparator comparator, boolean keepUpdated)	Xây dựng enumeration dùng để duyệt recordstore
void addRecordListener (RecordListener listener)	Add a listener to detect record store
void removeRecordListener (RecordListener listener)	Remove listener.

Chúng ta hãy cùng xem qua ví dụ đơn giản của việc đọc ghi record trong RecordStore.
Ví dụ: Đọc và ghi đối tượng string (ReadWrite.java)

```
/*-----------------------------------------------
* ReadWrite.java
*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
public class ReadWrite extends MIDlet {
        private RecordStore rs = null;
        static final String REC_STORE = "db_1";
        public ReadWrite() {
                openRecStore(); // Create the record store
                // Write a few records and read them back
                writeRecord("J2ME and MIDP");
                writeRecord("Wireless Technology");
                readRecords();
                closeRecStore(); // Close record store
                deleteRecStore(); // Remove the record store
        }
        public void destroyApp( boolean unconditional ) { }
        public void startApp() {
                // There is no user interface, go ahead and shutdown
                destroyApp(false);
                notifyDestroyed(); }
        public void pauseApp() { }
        public void openRecStore() {
                try {
                        // Create record store if it does not exist
                        rs = RecordStore.openRecordStore(REC_STORE, true );
                }
                catch (Exception e) {
                        db(e.toString());
                }
        }
        public void closeRecStore() {
                try {
                        rs.closeRecordStore();
                }
                catch (Exception e) {
                        db(e.toString());
                }
```

```
    }
    public void deleteRecStore() {
        if (RecordStore.listRecordStores() != null) {
            try {
                RecordStore.deleteRecordStore(REC_STORE);
            }
            catch (Exception e) {
                db(e.toString());
            }
         }
    }
    public void writeRecord(String str) {
        byte[] rec = str.getBytes();
        try {
            rs.addRecord(rec, 0, rec.length);
        }
        catch (Exception e) {
             db(e.toString());
        }
    }
    public void readRecords() {
        try {
            byte[] recData = new byte[50];
            int len;
            for (int i = 1; i <= rs.getNumRecords(); i++) {
               len = rs.getRecord( i, recData, 0 );
               System.out.println("Record #" + i + ": " +     new
            String(recData, 0, len));
               System.out.println("------------------------------");
            }
        }
        catch (Exception e) {
            db(e.toString());
        }
    }
    private void db(String str) {
        System.err.println("Msg: " + str); }
    }
}
```

Đây là output của ví dụ 1:



Hàm để mở một recordstore

```
public void openRecStore() {
        try {
                // Create record store if it does not exist
                rs = RecordStore.openRecordStore(REC_STORE, true );
        }
        catch (Exception e) {
                 db(e.toString());
        }
 }
```

Với tham số true, hàm sẽ tạo một RecordStore nếu nó chưa tồn tại. Trong hàm WriteRecord, trước khi lưu vào RecordStore, cần phải chuyển đổi kiểu string thành dãy byte:

```
byte[] rec = str.getBytes();
...
rs.addRecord(rec, 0, rec.length);
```

Trong hàm ReadRecord, chúng ta cũng cần đọc một dãy byte:

```
byte[] recData = new byte[50];
...
len = rs.getRecord( i, recData, 0 );
```

Cần lưu ý là trong ví dụ trên do biết trước kích thước của string nên khai báo dãy byte vừa đủ, trong thực tế ta nên kiểm tra kích thước của record để khai báo dãy byte cần thiết để tránh phát sinh lỗi, do đó hàm ReadRecord có thể sửa lại như sau:

```
for (int i = 1; i <= rs.getNumRecords(); i++) {
        if (rs.getRecordSize(i) > recData.length)
                recData = new byte[rs.getRecordSize(i)];
```

```
        len = rs.getRecord(i, recData, 0);
        System.out.println("Record #" + i + ": " + new String(recData, 0, len));
        System.out.println("----------------------------");
}
```

Nếu chỉ cần đọc ghi những đoạn text vào record, thì ví dụ trên là quá đủ. Tuy nhiên, thực tế là ta cần lưu những giá trị khác: String, int, boolean, v.v… Trong trường hợp này, chúng ta cần sử dụng stream để đọc và ghi record. Việc sử dụng stream giúp chúng ta linh động và nâng cao hiệu quả của việc đọc và ghi dữ liệu vào RecordStore. Chúng ta sử dụng nextRecord() để duyệt đến record sau đó, ngoài ra còn có previousRecord() giúp duyệt về record trước đó. Nếu muốn bắt đầu tại vị trí cuối cùng của recordstore ta chỉ cần gọi hàm previousRecord() ngay khi mở recordstore, nó sẽ trả về dòng cuối cùng.

RecordEnumeration có duy trì một index của các record. Khi recordstore có sự thay đổi thì RecordEnumeration có thể hoạt dộng không chính xác, do đó chúng ta cần phải gọi hàm reindex() mỗi khi recordstore có sự thay đổi.

RecordEnumeration API

RecordEnumeration Interface: javax.microedition.rms.RecordEnumeration	
Method	**Description**
int numRecords()	Số lượng record trong enumeration
byte[] nextRecord()	Record tiếp theo
int nextRecordId()	Record ID của record tiếp theo
byte[] previousRecord()	Record trước đó
int previousRecordId()	Record ID của record trước đó
boolean hasNextElement()	Kiểm tra enumeration có record kế tiếp
boolean hasPreviousElement()	Kiểm tra enumeration có record trước đó
void keepUpdated(boolean keepUpdated)	Đặt enumeration reindex sau khi co sựthay đổi
boolean isKeptUpdated()	Kiểm tra enumeration có tựđộng reindex()
void rebuild()	Tạo lại index
void reset()	Đưa enumeration về record đầu tiên
void destroy()	Giải phóng tài nguyên được sử dụng bởi enumeration

4. Sắp Xếp Các Record Với interface RecordComparator

Interface này giúp người lập trình so sánh hai Record theo một tiêu chí nào đó. Interface này định nghĩa phương thức compare với trị đầu vào là hai mảng các byte thể hiện hai Record cần so sánh. Phương thức này trả về các trị sau được định nghĩa trong interface:

EQUIVALENT: Nếu hai Record bằng nhau

FOLLOWS: Nếu Record thứ 1 đứng sau Record thứ 2

PRECEDES: Nếu Record thứ 1 đứng trước Record thứ 2

Do RecrdComparator là một interface nên khi sử dụng cần phải implements nó:

```
public class Comparator implements RecordComparator {
        public int compare(byte[] rec1, byte[] rec2) {
                String str1 = new String(rec1),
                        str2 = new String(rec2);
                int result = str1.compareTo(str2);
                if (result == 0) return RecordComparator.EQUIVALENT;
                else if (result < 0) return RecordComparator.PRECEDES;
                else return RecordComparator.FOLLOWS;
        }
}
```

Sau đó ta sử dụng lớp Comparator bằng cách gắn kết nó với RecordEnumeration:

```
// Create a new comparator for sorting
Comparator comp = new Comparator();
// Reference the comparator when creating the result set
RecordEnumeration re = rs.enumerateRecords(null,comp,false);
// Iterate through the sorted results while (re.hasNextElement()) {
String str = new String(re.nextRecord()); .
```

Enumeration sẽ sử dụng hàm compare trong class Comparator để sắp xếp các record trong RecordStore.

RecordComparator Interface: javax.microedition.rms.RecordComparator	
Method	**Description**
int compare(byte[] rec1, byte[] rec2)	So sánh để quyết định thứ tự sắp xếp

Ví dụ: chương trình sắp xếp cơ bản

```
/*-----------------------------------------------
* SimpleSort.java
*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
public class SimpleSort extends MIDlet {
        private RecordStore rs = null;
        static final String REC_STORE = "db_1";
        public SimpleSort() {
                openRecStore();     // Create the record store
                // Write a few records
                writeRecord("Sand Wedge");
                writeRecord("One Wood");
                writeRecord("Putter");
                writeRecord("Five Iron");
                // Read back with enumerator, sorting the results
                readRecords();
                closeRecStore(); // Close record store
                deleteRecStore(); // Remove the record store
        }
        public void destroyApp( boolean unconditional ) { }
        public void startApp() {
                // There is no user interface, go ahead and shutdown
                destroyApp(false);
                notifyDestroyed();
        }
        public void pauseApp() {}
        public void openRecStore() {
                try {
                        // Create record store if it does not exist
                        rs = RecordStore.openRecordStore(REC_STORE, true );
                }
                catch (Exception e) {
                        db(e.toString());
                }
        }
        public void closeRecStore() {
                try {
                        rs.closeRecordStore();
                }
```

```
        catch (Exception e) {
                db(e.toString());
        }
}
public void deleteRecStore() {
        if (RecordStore.listRecordStores() != null) {
                try {
                        RecordStore.deleteRecordStore(REC_STORE);
                }
                catch (Exception e) {
                        db(e.toString());
                }
        }
}
public void writeRecord(String str) {
        byte[] rec = str.getBytes();
        try {
                rs.addRecord(rec, 0, rec.length);
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void readRecords() {
        try {
                if (rs.getNumRecords() > 0) {
                        Comparator comp = new Comparator();
                        RecordEnumeration re = rs.enumerateRecords(null,comp,
                        false);     while (re.hasNextElement()) {
                                String str = new String(re.nextRecord());
                                System.out.println(str);
                                System.out.println("----------------------------");
                        }
                }
        }
        catch (Exception e) {
                db(e.toString());
        }
 }
private void db(String str) {
        System.err.println("Msg: " + str);
```

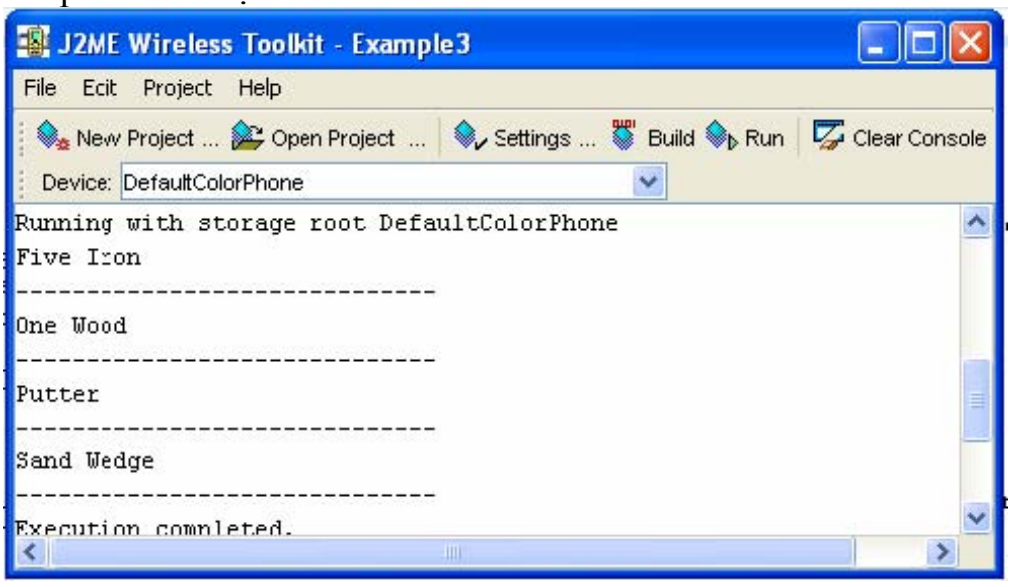
```
        }

}
class Comparator implements RecordComparator {
        public int compare(byte[] rec1, byte[] rec2) {
                String str1 = new String(rec1),
                        str2 = new String(rec2);
                int result = str1.compareTo(str2);
                if (result == 0) return RecordComparator.EQUIVALENT;
                else if (result < 0)    return RecordComparator.PRECEDES;
                else   return RecordComparator.FOLLOWS;
        }
}
```

Trong đoạn code trên trong hàm readRecord(), khi tạo Enumeration ta đã tham chiếu đến đối tượng comp của lớp Comparator

```
Comparator comp = new Comparator();
RecordEnumeration re = rs.enumerateRecords(null, comp, false);
while (re.hasNextElement()) { ... }
```

Khi enumerator tạo index cho RecordStore nó sẽ sử dụng hàm compare() ở trên để sắp xếp các record.

Output của vi dụ:



Ví dụ trên đúng trong trường hợp dữ liệu lưu vào record là dạng text, nếu quay lại ta đã ghi nhiều kiểu dữ liệu vào trong một record:

```
// Write Java data types to stream
strmDataType.writeUTF("Text 1");
```

strmDataType.writeBoolean(true);

strmDataType.writeInt(1);

thì các kiểu dữ liệu trên sẽ được lưu vào một stream ở dạng binary. Sau đó các stream này sẽ được chuyển thành mảng và đưa vào recordstore:

// Get stream data into an array

record = strmBytes.toByteArray();

// Write the array to a record

rs.addRecord(record, 0, record.length);

Đoạn code trong ví dụ trên sẽ chạy sai khi áp dụng với kiểu dữ liệu binary. Để giải quyết, ta cần phải viết lại hàm compare() thự c hiện chức năng chuyển đổi chuỗi byte và sắp xếp đúng kiểu dữ liệu.

Trong thực tế, chúng ta cần phải lưu nhiều trường dữ liệu trong một record như trong ví dụ 2 (lưu dữ liệu kiểu String, boolean, integer). Trong trường hợp này sẽ có nhiều lựa chọn đểsắp xếp các record, và việc lựa chọn này tùy thuộc vào ứng dụng.

Trong 2 ví dụ sau đây sẽ thực thi interface RecordComparator để sắp xếp record chứa nhiều kiểu dữ liệu. Những ví dụ này sẽ sử dụng cùng dữ liệu đầu vào, tuy nhiên ví dụ 4 sẽ sắp xếp dựa vào kiểu String, trong khi ví dụ 5 sẽ sắp xếp dựa vào kiểu integer. Đây là dữ liệu sẽ lưu vào recordstore:

String[] pets = {"duke", "tiger", "spike", "beauregard"};

boolean[] dog = {true, false, true, true};

int[] rank = {3, 0, 1, 2};

Khi lưu vào recordstore sẽ có dạng như sau:

Record #1

"duke" true 3

Record #2

"tiger" false 0

Record #3

"spike" true 1

Record #4

"beauregard" true 2

Đây là lý do ví dụ trên không đáp ứng được yêu cầu, do dữ liệu lưu vào không còn là dạng text, và hàm String.CompareTo() trên nội dung của record không thể sắp xếp dữ liệu theo mong muốn. Do đó cần phải lấy ra từ mỗi record trường dữ liệu mà ta muốn sắp xếp.

Ví dụ 5: integer sort

```
/*-------------------------------------------------
* IntSort.java
*

------------*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
public class IntSort extends MIDlet {
        private RecordStore rs = null; // Record store
        static final String REC_STORE = "db_4"; // Name of record store
         public IntSort() {
                openRecStore(); // Create the record store
                writeTestData(); // Write a series of records
                readStream(); // Read back the records
                closeRecStore(); // Close record store
                deleteRecStore(); // Remove the record store
        }
        public void destroyApp( boolean unconditional ) {}
        public void startApp() {
                // There is no user interface, go ahead and shutdown
                destroyApp(false);
                notifyDestroyed();
        }
        public void pauseApp() {}
        public void openRecStore() {
                try {
```

```
                // Create record store if it does not exist
                rs = RecordStore.openRecordStore(REC_STORE, true );
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void closeRecStore() {
        try {
                rs.closeRecordStore();
        }
        catch (Exception e) {
                 db(e.toString());
        }
}
public void deleteRecStore() {
        if (RecordStore.listRecordStores() != null) {
                try {
                        RecordStore.deleteRecordStore(REC_STORE);
                }
                catch (Exception e) {
                        db(e.toString());
                }
        }
}
public void writeTestData() {
        String[] pets = {"duke", "tiger", "spike", "beauregard"};
        boolean[] dog = {true, false, true, true};
        int[] rank = {3, 0, 1, 2};
        writeStream(pets, dog, rank);
}
public void writeStream(String[] sData, boolean[] bData,int[] iData) {
        try {
                // Write data into an internal byte array
                ByteArrayOutputStream strmBytes = new
        ByteArrayOutputStream();
                // Write Java data types into the above byte array
                DataOutputStream strmDataType =    new
        DataOutputStream(strmBytes);
                byte[] record; for (int i = 0; i < sData.length; i++) {
                        // Write Java data types
```

```
                    strmDataType.writeUTF(sData[i]);
                    strmDataType.writeBoolean(bData[i]);
                    strmDataType.writeInt(iData[i]);
                    // Clear any buffered data
                    strmDataType.flush();
                    // Get stream data into byte array and write record
                    record = strmBytes.toByteArray();
                    rs.addRecord(record, 0, record.length);
                    // Toss any data in the internal array so writes
                    // starts at beginning (of the internal array)
                    strmBytes.reset();
                }
                strmBytes.close();
                strmDataType.close();
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void readStream() {
        try {
                byte[] recData = new byte[50];
                // Read from the specified byte array
                ByteArrayInputStream strmBytes =    new
        ByteArrayInputStream(recData);
                // Read Java data types from the above byte array
                DataInputStream strmDataType =    new
        DataInputStream(strmBytes);
                if (rs.getNumRecords() > 0) {
                        ComparatorInt comp = new ComparatorInt();
                        int i = 1;
                        RecordEnumeration re = rs.enumerateRecords(null,comp,
                        false);     while (re.hasNextElement()) {
                                // Get data into the byte array
                                rs.getRecord(re.nextRecordId(), recData, 0);
                                // Read back the data types
                                System.out.println("Record #" + i++);
                                System.out.println("Name: " +
                                strmDataType.readUTF());
                                System.out.println("Dog: " +
                                strmDataType.readBoolean());
                                System.out.println("Rank: " +
```

```
                                strmDataType.readInt());      System.out.println("------
                                ----------");
                                // Reset so read starts at beginning of array
                                strmBytes.reset();
                        }
                        comp.compareIntClose();     // Free enumerator
                re.destroy();
                }
                strmBytes.close();
                strmDataType.close();
        }
        catch (Exception e) {
                db(e.toString());
        }
    }
    private void db(String str) {
        System.err.println("Msg: " + str);
    }
}

class ComparatorInt implements RecordComparator {
    private byte[] recData = new byte[10];
    // Read from a specified byte array
    private ByteArrayInputStream strmBytes = null;
    // Read Java data types from the above byte array
    private DataInputStream strmDataType = null;
    public void compareIntClose() {
        try {
            if (strmBytes != null)    strmBytes.close();
            if (strmDataType != null) strmDataType.close();
        }
        catch (Exception e) {}
    }
    public int compare(byte[] rec1, byte[] rec2) {
        int x1, x2;
        try {
            // If either record is larger than our buffer, reallocate
            int maxsize = Math.max(rec1.length, rec2.length);
            if (maxsize > recData.length)    recData = new byte[maxsize];
            // Read record #1
            // We want the integer from the record, which is
            // the last "field" thus we must read the String
```

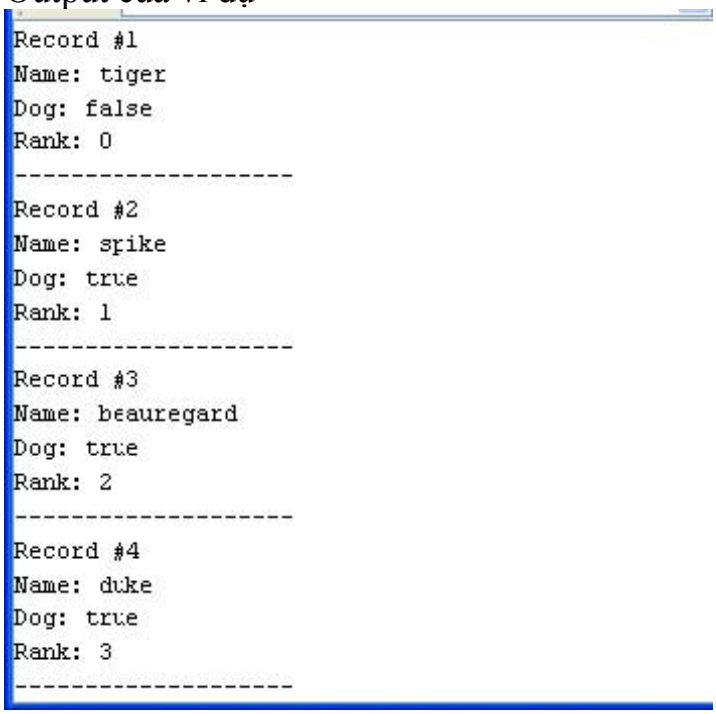
```
                    // and boolean to get to the integer
                    strmBytes = new ByteArrayInputStream(rec1);
                    strmDataType = new DataInputStream(strmBytes);
                    strmDataType.readUTF();
                    strmDataType.readBoolean();
                    x1 = strmDataType.readInt();
                    // Here's our data
                    // Read record #2
                    strmBytes = new ByteArrayInputStream(rec2);
                    strmDataType = new DataInputStream(strmBytes);
                    strmDataType.readUTF();
                    strmDataType.readBoolean();
                    x2 = strmDataType.readInt();
                    // Here's our data
                    // Compare record #1 and #2
                    if (x1 == x2)   return RecordComparator.EQUIVALENT;
                    else if (x1 < x2)    return RecordComparator.PRECEDES;
                    else   return RecordComparator.FOLLOWS;
                }
                catch (Exception e) {
                    return RecordComparator.EQUIVALENT;
                }
        }
}
```

Trong ví dụ này tiêu chí sắp xếp là theo kiểu integer, do đó trước hết ta phải lấy dữ liệu trong dãy byte. Tuy nhiên, có một lưu ý là do dữ liệu ta cần lấy nằm cuối cùng trong dãy byte do đó ra cần phải đọc theo thứ tự, tức là phải đọc kiểu String, boolean rồi mới đến integer:

```
// Read record #1
// We want the integer from the record, which is
// the last "field" thus we must read the String
// and boolean to get to the integer
. . .
strmDataType.readUTF();
strmDataType.readBoolean();
x1 = strmDataType.readInt();
// Here's our data
// Read record #2
. . .
strmDataType.readUTF();
strmDataType.readBoolean();
```

```
x2 = strmDataType.readInt();
// Here's our data
// Compare record #1 and #2
 . . .
```

Output của ví dụ



5. Tìm Kiếm Với Bộ Lọc RecordFilter

Ngoài việc sắp xếp các record (sử dụng RecordComparator), enumerator còn cung cấp cơ chế lọc (tìm kiếm các record thỏa mãn một điều kiện nào đó). Khi sử dụng RecordComparator tất cả các record trong RecordStore đều được lưu trong một result set. Nhưng khi dùng RecordFilter, chỉ có những thỏa mãn điều kiện mới có trong enumerator result set.

```
class SearchFilter implements RecordFilter {
        private String searchText = null;
        public SearchFilter(String searchText) {
                // This is the text to search for
                this.searchText = searchText.toLowerCase();
        }
        public boolean matches(byte[] candidate) {
                String str = new String(candidate).toLowerCase();
                // Look for a match
                if (searchText != null && str.indexOf(searchText) != -1) return true;
                else return false;
        }
}
```

Trên đây là một class đơn giản thực thi interface RecordFilter. Class này sẽđược gắn với một enumerator, và khi đó enumerator sẽ dùng hàm matches() duyệt hết recordstore lấy ra những record cần tìm:

```
// Create a new search filter
SearchFilter search = new SearchFilter("search text");
// Reference the filter when creating the result set
RecordEnumeration re = rs.enumerateRecords(search,null,false);
// If there is at least one record in result set, a match was found
if (re.numRecords() > 0)
// Do something
```

RecordFilter Interface: javax.microedition.rms.RecordFilter	
Method	**Description**
boolean matches(byte[] candidate)	Tìm kiếm record thỏa mãn một điều kiện nào đó

Sau đây ta sẽ xem qua chương trình tìm kiếm đơn giản sử dụng interface RecordFilter:
Ví dụ

```
/*----------------------------------------------
* SimpleSearch.java
*
*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
public class SimpleSearch extends MIDlet implements CommandListener {
        private Display display; // Reference to Display object
        private Form fmMain; // The main form private
        StringItem siMatch; // The matching text, if any
        private Command cmFind; // Command to search record store
        private Command cmExit; // Command to insert items
        private TextField tfFind; // Search text as requested by user
        private RecordStore rs = null; // Record store
        static final String REC_STORE = "db_6"; // Name of record store
        public SimpleSearch() {
                display = Display.getDisplay(this);
                tfFind = new TextField("Find", "", 10, TextField.ANY);
                siMatch = new StringItem(null, null);
                cmExit = new Command("Exit", Command.EXIT, 1);
```

```
        cmFind = new Command("Find", Command.SCREEN, 2);
        // Create the form, add commands
        fmMain = new Form("Record Search");
        fmMain.addCommand(cmExit);
        fmMain.addCommand(cmFind);
        // Append textfield and stringItem
        fmMain.append(tfFind);
        fmMain.append(siMatch);
        // Capture events
        fmMain.setCommandListener(this);
        //------------------------------- // Open and write to record store   //----------
        ---------------------- openRecStore();
        // Create the record store
        writeTestData();
        // Write a series of records
}
public void destroyApp( boolean unconditional ) {
        closeRecStore(); // Close record store
        deleteRecStore();
}
public void startApp() {
        display.setCurrent(fmMain);
}
public void pauseApp() {}
public void openRecStore() {
        try {
                // Create record store if it does not exist
                rs = RecordStore.openRecordStore(REC_STORE, true );
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void closeRecStore()  {
        try {
        rs.closeRecordStore();
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void deleteRecStore() {
```

```
        if (RecordStore.listRecordStores() != null) {
                try {
                        RecordStore.deleteRecordStore(REC_STORE);
                }
                catch (Exception e) {
                        db(e.toString());
                }
        }
}
public void writeTestData() {
        String[] golfClubs = { "Wedge...good from the sand trap", "Truong dai
        hoc Cong nghe ", "Putter...only on the green", "Hoc mon LTUDM rat bo
        ich!"};   writeRecords(golfClubs);
}
public void writeRecords(String[] sData) {
        byte[] record;
        try {
                // Only add the records once
                if (rs.getNumRecords() > 0) return;
                for (int i = 0; i < sData.length; i++) {
                        record = sData[i].getBytes();
                        rs.addRecord(record, 0, record.length);
                }
        }
        catch (Exception e) {
                db(e.toString());
        }
}
private void searchRecordStore() {
        try {
                // Record store is not empty
                if (rs.getNumRecords() > 0) {
                        // Setup the search filter with the user requested text
                        SearchFilter search =     new
                        SearchFilter(tfFind.getString());     RecordEnumeration re
                        =     rs.enumerateRecords(search, null, false);
                        // A match was found using the filter
                        if (re.numRecords() > 0)
                                // Show match in the stringItem on the form
                                siMatch.setText(new String(re.nextRecord()));
                re.destroy(); // Free enumerator
```

```
                }
            }
            catch (Exception e) {
                db(e.toString());
            }
        }
        public void commandAction(Command c, Displayable s) {
            if (c == cmFind) {
                searchRecordStore();
            }
            else if (c == cmExit) {
                destroyApp(false);
                notifyDestroyed();
            }
        }
        private void db(String str) {
            System.err.println("Msg: " + str);
        }
}

class SearchFilter implements RecordFilter {
        private String searchText = null;
        public SearchFilter(String searchText) {
            // This is the text to search for
            this.searchText = searchText.toLowerCase();
        }
        public boolean matches(byte[] candidate) {
            String str = new String(candidate).toLowerCase();
            // Look for a match
            if (searchText != null && str.indexOf(searchText) != -1) return true;
            else   return false;
        }
}
```

Sau khi viết class SearchFilter, ta tạo một instance search, khi khai báo class RecordEnumeration sẽ tham chiếu đến instance trên. Khi đó chỉ có những record thỏa mãn điều kiện (trong hàm matches()) mới hiển thị trong result set:

```
        // Setup the search filter with the user requested text
        SearchFilter search = new SearchFilter(tfFind.getString());
        RecordEnumeration re =rs.enumerateRecords(search,null,false);
        // A match was found using the filter
        if (re.numRecords() > 0) siMatch.setText(new String(re.nextRecord()));
```

Output:

6. Nhận Biết Thay Đổi Với RecordListener

Để phát hiện các thay đổi cũng như thêm vào các Record trong RecordStore, RMS cung cấp giao diện RecordListener. Giao diện này định nghĩa 3 phương thức, các phương thức có 2 trị vào là một đối tượng kiểu RecordStore và một số int chứa recordID. Các phương thức đó là:

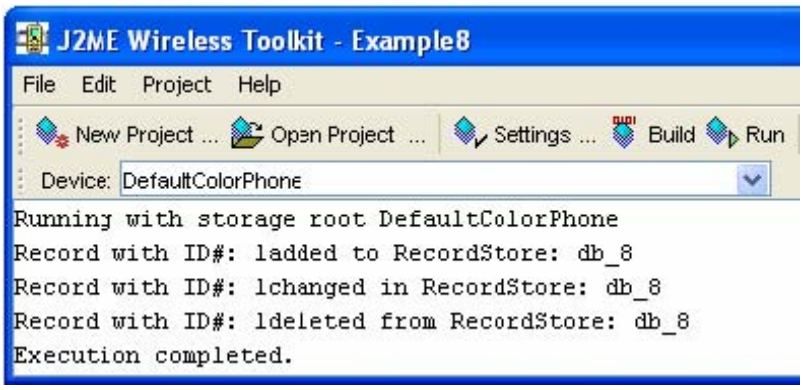
RecordListener Interface: javax.microedition.rms.RecordListener	
Method	**Description**
void recordAdded(RecordStore recordStore, int recordId)	Được gọi khi thêm 1 record
void recordChanged(RecordStore recordStore, int recordId)	Được gọi khi record bi thay đổi
void recordDeleted(RecordStore recordStore, int recordId)	Được gọi khi record bị xóa

Ví dụ : sử dụng RecordListener

```
/*----------------------------------------------
* RmsListener.java
*

*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
public class RmsListener extends MIDlet {
	private RecordStore rs = null;
	static final String REC_STORE = "db_8";
	public RmsListener() {
		// Open record store and add listener
		openRecStore();
		rs.addRecordListener(new TestRecordListener());
		// Initiate actions that will wake up the listener
		writeRecord("J2ME and MIDP");
		updateRecord("MIDP and J2ME");
		deleteRecord();
		closeRecStore();
		// Close record store
		deleteRecStore();
		// Remove the record store
	}
	public void destroyApp( boolean unconditional ) {}
	public void startApp() {
		// There is no user interface, go ahead and shutdown
		destroyApp(false);
		notifyDestroyed();
	}
	public void pauseApp() {}
	public void openRecStore() {
		try {
			// Create record store if it does not exist
			rs = RecordStore.openRecordStore(REC_STORE, true );
		}
		catch (Exception e) {
			db(e.toString());
		}
	}
	public void closeRecStore() {
```

```
        try {
                rs.closeRecordStore();
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void deleteRecStore() {
        if (RecordStore.listRecordStores() != null) {
                try {
                        RecordStore.deleteRecordStore(REC_STORE);
                }
                catch (Exception e) {
                        db(e.toString());
                }
        }
}
public void writeRecord(String str) {
        byte[] rec = str.getBytes();
        try {
                rs.addRecord(rec, 0, rec.length);
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void updateRecord(String str) {
        try {
                rs.setRecord(1, str.getBytes(), 0, str.length());
        }
        catch (Exception e) {
                db(e.toString());
        }
}
public void deleteRecord() {
        try  {
                rs.deleteRecord(1);
        }
        catch (Exception e) {
                db(e.toString());
```

```
            }
        }
        public void db(String str) {
            System.err.println("Msg: " + str);
            }
}
class TestRecordListener implements RecordListener {
        public void recordAdded(RecordStore recordStore, int recordId) {
            try {
                System.out.println("Record with ID#: " + recordId + "added to
        RecordStore: "
                + recordStore.getName());
            }
            catch (Exception e) {
                System.err.println(e);
            }
        }
        public void recordDeleted(RecordStore recordStore, int recordId) {
            try {
                System.out.println("Record with ID#: " + recordId + "deleted from
                RecordStore: " + recordStore.getName());
            }
            catch (Exception e) {
                System.err.println(e);
            }
        }
        public void recordChanged(RecordStore recordStore, int recordId) {
            try{
                System.out.println("Record with ID#: " + recordId +
                   "changed in RecordStore: "
                recordStore.getName());
            }
            catch (Exception e) {
                System.err.println(e);
            }
        }
}
```



7. Các Ngoại Lệ Phát Sinh Trong RMS

Các phương thức trong API của RMS ngoài việc phát sinh các ngoại lệ thông thường đến môi trường chạy (runtime enviroment). RMS còn định nghĩa thêm các ngoại lệ trong gói javax.microedition.rms như sau:

InvalidRecordIDException: Ngoại lệ này phát sinh ra khi không thể thao tác trên Record vì RecordID không thích hợp.

RecordStoreFullException: Ngoại lệ này phát sinh ra khi không còn đủ vùng nhớ.

RecordStoreNotFoundException: Ngoại lệ này phát sinh ra khi mở một RecordStore không tồn tại.

RecordStoreNotOpenException: Ngoại lệ này phát sinh ra khi thao tác trên một RecordStore đã bịđóng.

RecordStoreException: Đây là lớp cha của 4 lớp trên, ngoại lệ này mô tả lỗi chung nhất trong quá trình thao tác với RMS.

Chương IV: KHUNG KẾT NỐI CHUNG (Generic Connection Framework - GCF)

Trong phiên bản J2SE, hỗ trợ các giao thức kết nối mạng có các gói java.io và java.net với tổng dung lượng hơn 200KB bao gồm hơn 100 lớp và giao diện. Qaủ thật với bộ nhớ nhỏ bé và hạn chế trong xử lý, việc đưa những gói này vào trong ứng dụng viết bằng J2ME là một điều hoàn toàn không khả thi. Chính vì vậy, khi mở rộng phạm vi hỗ trợ giao thức mạng và hệ thống tập tin, ngươiừ ta không dùng lại các lớp của J2SE mà xây dựng một khái niệm mới được gọi là Khung kết nối chung (Generic Connection Framework - GCF).

GCF là một tập hợp các lớp và giao diện được thiết kế nhằm tọa thuận tiện cho việc truy xuất đến các hệ thống lưu trữ và kết nối mạng. Mục tiêu của GCF không phải là tạo ra một tập các lớp mới hoàn toàn mà nó cung cấp một tập con của J2SE một cách có chọn lọc. Tập con này được giới hạn và tối ưu để phù hợp với những ràng buộc và khác biệt của những thiết bị di động.

1. Cây phân cấp Connection

Khi đưa ra khái niệm cây phân cấp, người ta chủ ý tạo ra một lớp có khả năng mở mọi loại kết nối bao gồm: file, http, datagram, … Tên của lớp này là Connector. Như vậy nếu sử dụng Connector để mở kết nối, chúng ta chỉ cần gọi một phương thức open có định dạng như sau:

Connector.Open(“protocol:address; parameter”)

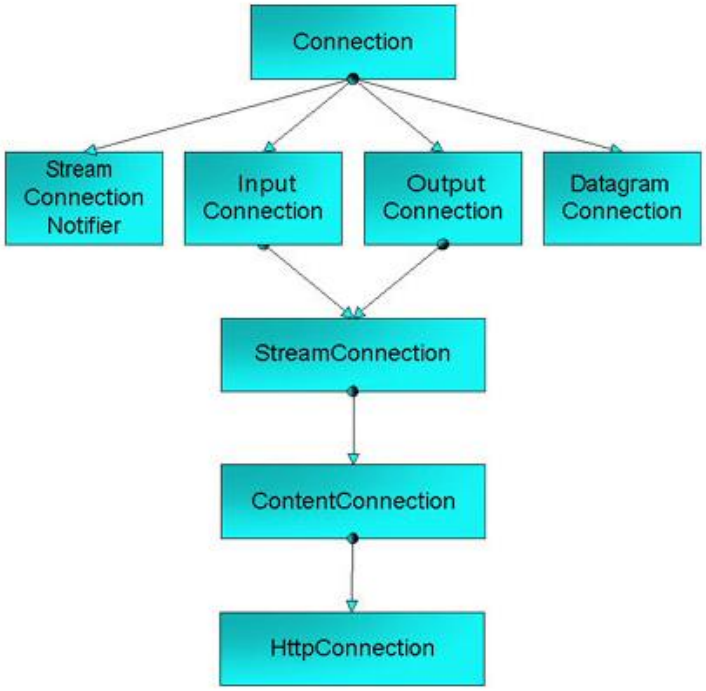
Cơ chế mà GCF dùng để mở nhiều loại giao tiếp chỉ bằng một phương thức chung duy nhất này đã chứng minh tính uyển chuyển của GCF. Cơ chế này hoạt động như sau:

Trong thời gian thực thi, mỗi khi có yêu cầu mở một giao thức, Connector sẽ tìm đến lớp tương ứng cài đặt giao thức ấy. Quá trình tìm kiếm này được thực hiện thông qua phương thức Class.forName(). Ví dụ như để yêu cầu mở kết nối HTTP trong J2ME, yêu cầu đó sẽ được viết như sau:

Class.forName(“com.sun.midp.io.j2me.http.Protocol”);

Khi tìm thấy lớp tương ứng, Class.forName() sẽ trả về một đối tượng có cài đặt giao diện Connection (trong đó lớp Connector và giao diện Connection đã được định nghĩa sẵn trong CLDC)

Sau đây là cây phân cấp Connection, nó bao gồm các lớp mà mỗi lớp được định nghĩa như là một giao diện



Cây phân cấp Connection - Khung kết nối chung

Trong kiến trúc của cây phân cấp, cài đặt thật sự của ác giao thức đều nằm ở mức hiện trạng. Trong MIDP 1.0, HttpConnection hỗ trợo một tâpj con HTTP phiên bản 1.0. Do đó khi lớp này mở rộng ContentConnection, nó đã được cung cấp sẵn hơn 20 phương thức chuyên biệt để giao tiếp thông qua giao thức HTTP.

Mặc dù DatagramConnection cũng xuất hiện trong cây phân cấp nhưng người ta không bắt buộc cài đặt MIDP để hỗ trợ giao thức này.

2. Kết nối HTTP

HTTP là giao thức duy nhất chắc chắn được hỗ trợ bởi MIDP 1.0. Chúng ta có thể giao tiếp với máy chủ hay bất kỳ thiết bị từ xa nào có hỗ trợ giao thức này nhờ vào lớp HttpConnection. Lớp Connector cung cấp cho người dùng bảy phương thức để tạo kết nối tới máy chủ. Ba phương thức trong số đó là các biến thể của phương thức open(). Các phương thức này được mô tả trong bảng sau:

Các phương thức của lớp javax.microedition.io.Connector	
Phương thức	**Mô tả**
static Connection open(String name)	Tạo một kết nối có chế độ READ_WRITE
static Connection open(String name, int mode)	Tạo một kết nối với chế độ được chỉ định
static Connection open(String name, int mode, boolean timeouts)	Tạo một kết nối với chế độ được chỉ định, thêm ngoại lệ time out
static InputStream openInputStream(String name)	Tạo kết nối luồng nhập
static OutputStream openOutputStream(String name)	Tạo kết nối luồng xuất
static DataInputStream openDataInputStream(String name)	Tạo kết nối luồng nhập kiểu DataInputStream
static DataOutputStream openDataOutputStream(String name)	Tạo kết nối luồng xuất kiểu DataOutputStream

Dưới đây là đoạn code mở kết nối thông qua stream

```
// Create a ContentConnection
String url = "http://www.corej2me.com";
ContentConnection connection = (ContentConnection) Connector.open(url);
// With the connection, open a stream
InputStream iStrm = connection.openInputStream();
// ContentConnection includes a length method
int length = (int) connection.getLength();
if (length != -1) {
        byte imageData[] = new byte[length];
        // Read the data into an array
        iStrm.read(imageData);
}
```

Thật ra chúng ta có thể tọa một kết nối InputStream mà không cần sự có mặt của ContentConnection. Tuy nhiên, phương pháp này có hạn chế là không cung cấp phương thức để xác định chiều dài dữ liệu

Dưới đây là cách mở một kết nối dạng HttpConnection:

```
String url = "http://www.corej2me.com/midbook_v1e1/ch14/duke.png";
HttpConnection http = (HttpConnection) Connector.open(url);
```

Sau khi được mở, kết nối này cung cấp truy xuất đến rất nhiều loại luồng mà InputStream và DataInputStream là hai trong số đó. Tuy nhiên thế mạnh thực sự của kết nối HttpConnection lại nằm ở chỗ nó có khả năng giúp cho lập trình viên loại bỏ các gánh nặng của các câu lệnh HTTP.

Dưới đây là ví dụ đơn giản, đầu tiên MIDlet sẽ download và hiển thị hình ảnh đã tải về. MIDlet sẽ dử dụng ByteArrayOutputStream để chứa dữ liệu tải về bởi vì ta không dùng ContentConnection nên không thể biết kích cỡ dữ liệu tải về

```
/*-------------------------------------------
* DownloadImage.java *
*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
public class DownloadImage extends MIDlet implements CommandListener {
    private Display display;
    private TextBox tbMain;
    private Form fmViewPng;
    private Command cmExit;
    private Command cmView;
    private Command cmBack;
    public DownloadImage() {
            display = Display.getDisplay(this);
            // Create the textbox, allow maximum of 50 characters
            tbMain = new TextBox("Enter url", "http://localhost/intel.png", 55, 0);
            // Create commands and add to textbox
            cmExit = new Command("Exit", Command.EXIT, 1);
            cmView = new Command("View", Command.SCREEN, 2);
            tbMain.addCommand(cmExit);
            tbMain.addCommand(cmView );
            // Set up a listener for textbox
            tbMain.setCommandListener(this);
            // Create the form that will hold the image
            fmViewPng = new Form("");
            // Create commands and add to form
            cmBack = new Command("Back", Command.BACK, 1);
            fmViewPng.addCommand(cmBack);
            // Set up a listener for form
            fmViewPng.setCommandListener(this);
    }
    public void startApp() {
            display.setCurrent(tbMain);
    }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable s) {
```

```
        // If the Command button pressed was "Exit"
        if (c == cmExit) {
                destroyApp(false);
        notifyDestroyed();
        }
        else if (c == cmView) {
                // Download image and place on the form
                try {
                  Image im;
                        if ((im = getImage(tbMain.getString())) != null) {
                                ImageItem ii = new ImageItem(null, im,
                                        ImageItem.LAYOUT_DEFAULT, null);
                          // If there is already an image, set (replace) it
                          if (fmViewPng.size() != 0) fmViewPng.set(0, ii);
                          else // Append the image to the empty form
                               fmViewPng.append(ii);
                         }
                         else fmViewPng.append("Unsuccessful download.");
                // Display the form with image
                display.setCurrent(fmViewPng);
                }
                catch (Exception e) {
                        System.err.println("Msg: " + e.toString());
                }
        }
        else if (c == cmBack) {
                display.setCurrent(tbMain);
        }
}

private Image getImage(String url) throws IOException {
        InputStream iStrm = (InputStream) Connector.openInputStream(url);
        Image im = null;
        try {
                ByteArrayOutputStream bStrm = new ByteArrayOutputStream();
                int ch;
                while ((ch = iStrm.read()) != -1) bStrm.write(ch);
                // Place into image array
                byte imageData[] = bStrm.toByteArray();
                // Create the image from the byte array
                im = Image.createImage(imageData, 0, imageData.length);
        }
```

```
            finally {
                // Clean up
                if (iStrm != null)
                iStrm.close();
            }
            return (im == null ? null : im);
        }
}
```

Một textbox sẽ cho phép nhập địa chỉ URL

Sau khi tải về, hình ảnh sẽđược hiển thị

3. Client Request và Server Response

Cả HTTP và HTTPS đều gửi request và response. Máy client gửi request, còn server sẽ trả về response. Client request bao gồm 3 phần sau:

Request method

Header

Body

Request method định nghĩa cách mà dữ liệu sẽ được gửi đến server. Có 3 phương thức được cung cấp sẵn là GET, POST, HEADER. Khi sử dụng Get, dữ liệu cần request sẽ nằm trong URL. Với Post dữ liệu gửi từ client sẽđược phân thành các stream riêng biệt. Trong khi đó, Header sẽ không gửi dữ liệu yêu cầu lên server, thay vào đó header chỉ request những meta information về server. GET và POST là hai phương thức request khá giống nhau, tuy nhiên do GET gửi dữ liệu thông qua URL nên sẽ bị giới hạn, còn POST sử dụng những stream riêng biệt nên sẽ khắc phục được hạn chế này.

Ví dụ về việc mở HTTP Connection thông qua GET

```
String url = "http://www.corej2me.com?size=large";
HttpConnection http = null;
http = (HttpConnection) Connector.open(url);
http.setRequestMethod(HttpConnection.GET);
```

Những Header field sẽ cho phép ta truyền các tham số từ client đến server. Các header field thường dùng là If-Modified-Since, Accept, and User Agent. Bạn có thểđặt các field này thông qua phương thức setRequestProperty(). Dưới đây là ví dụ dùng setRequestProperty(), chỉ có những dữ liệu thay đổi sau ngày 1 tháng 1 năm 2005 mới được gửi về từ server:

```
String url = "http://www.corej2me.com\somefile.txt";
HttpConnection http = null;
http = (HttpConnection) Connector.open(url);
http.setRequestMethod(HttpConnection.GET);
// Set header field as key-value pair
```

```
http.setRequestProperty("If-Modified-Since", "Sat, 1 Jan 2005 12:00:00
GMT");
```

Body chứa nội dung mà bạn muốn gửi lên server. Ví dụ về sử dụng POST và gửi dữ liệu từclient thông qua stream:

```
String url = “http://www.corej2me.com”;
tmp = "test data here";
OutputStream ostrm = null;
HttpConnection http = null;
http = (HttpConnection) Connector.open(url);
http.setRequestMethod(HttpConnection.POST);
// Send client body
ostrm = http.openOutputStream();
byte bytes[] = tmp.getBytes();
for(int i = 0; i < bytes.length; i++) {
    os.write(bytes[i]);
}
os.flush();
```

Sau khi nhận được và sử lý yêu cầu từ phía client, server sẽ đóng gói và gửi về phía client. Cũng như client request, server cũng gồm 3 phần sau:

Status line

Header

Body

Status line sẽ thông báo cho client kết quả của request mà client gửi cho server. HTTP phân loại status line thành các nhóm sau đây:

1xx is informational

2xx is success

3xx is redirection

4xx is client error

5xx is server error

Status line bao gồm version của HTTP trên server, status code, và đoạn text đại diện cho status code.

Ví dụ: "HTTP/1.1 200 OK" "HTTP/1.1 400 Bad Request" "HTTP/1.1 500 Internal Server Error"

Header.

Không giống như header của client, server có thể gửi data thông qua header.

Sau đây là những phương thức dùng để lấy thông tin Header mà server gửi về:

String getHeaderField(int n) *Get header field value looking up by index*
String getHeaderField(String name) *Get header field value looking up by name*
String getHeaderFieldKey(int n) *Get header field key using index*

Server có thể trả về nhiều Header field. Trong trường hợp này, phương thức đầu tiên sẽ cho lấy header field thông qua index của nó. Còn phương thức thứ hai lấy nội dung header field dựa vào tên của header field. Còn nếu muốn biết tên (key) của header field, có thể dùng phương thức thứ 3 ở trên.

Sau đây là ví dụ về 3 phương thức trên, trong trường hợp server gửi về chuỗi "content-type=text/plain".

Method	Return value
http.getHeaderField(0)	"text-plain"
http.getHeaderField("content-type")	"text-plain"
http.getHeaderFieldKey(0)	"content-type"

Body: Cũng giống như client, server gửi hầu hết những thông tin trong phần body cho client. Client dùng input stream đểđọc kết quả trả về từ server.

The HttpConnection API

Như đã đề cập ở trên, ta sẽ sử dụng HttpConnection API để thiết lập kết nối trong MIDP. Dưới đây là những API trong HttpConnection:

Method	**Description**
long getDate()	Get header field date
long getExpiration()	Gets header field expiration

String getFile()	Gets filename from the URL
int getHeaderField(int n)	Gets header field value looking up by index
String getHeaderField(String name)	Gets header field value looking up by name
long getHeaderFieldDate(String name, long def)	Gets named field as a long (representing the date)
int getHeaderFieldInt(String name, int def)	Gets named field as an integer
String getHeaderFieldKey(int n)	Gets header field key using index
String getHost()	Gets host from the URL
long getLastModified()	Gets last-modified field value
String getPort()	Gets port from the URL
String getProtocol()	Gets protocol from the URL
String getQuery()	Gets the query string (only valid with GET request)
String getRef()	Gets the reference portion of URL
String getRequestMethod()	Gets the current setting of the request method (GET, POST or HEAD)
String getRequestProperty(String key)	Gets the current setting of a request property
int getResponseCode()	Gets the response code (numeric value)
String getResponseMessage()	Gets the response message (text value)
String getURL()	Gets the entire URL
void setRequestMethod(String method)	Sets the request method (GET, POST or HEAD)
void setRequestProperty(String key, String value)	Sets a request property (header information)

Chương V: TỔNG KẾT

Hiện nay, lập trình trên điện thoại di động là một lĩnh vực mới đang thu hút nhiều lập trình viên. Việc xây dựng các ứng dụng trên thiết bị các thiết bị nói chung và trên điện thoại di động nói riêng là rất cần thiết do sự phát triển của công nghệ di động. Trong phạm vi đề tài, em chỉ trình bày những phần cơ bản nhất về công nghệ J2ME và kỹ thuật lập trình cho điện thoại di động. Những phần này đã được nghiên cứu, tìm hiểu qua quá trình học tập cũng như làm việc. Hi vọng đề tài này sẽ trở thành một công cụ tham khảo có ích cho những người đang tham gia tìm hiểu về công nghệ J2ME.

Tuy vậy, do những hạn chế về trình độ cũng như thiết bị nên em không thể tránh khỏi những vướng mắc và sai sót trong quá trình tìm hiểu, nghiên cứu. Em rất mong được sự đánh giá và chỉnh sửa của các thầy hướng dẫn cũng như các bạn sinh viên đọc qua tài liệu này. Em xin chân thành cảm ơn.

Hà nội, ngày 26/11/2007

Bùi Duy Thành

Tài liệu tham khảo

[1]. Phương Lan, *Java tập 3*, NXB Lao động Xã hội, 2006.

[2]. Nguyễn Thị Bích Ngà, *Nền tảng công nghệ J2Me & MDP*, NXB Giao thông Vận tải, 2006.

[3]. Nguyễn Hữu Mai, *Tổng quan về J2ME, javavietnam.org*, 2004.

[4]. Lê Ngọc Quốc Khánh, *Phát triển ứng dụng J2ME và J2ME Wireless Toolkit*, 2004.

[5]. John W. Muchow, *Core J2ME™ Technology & MIDP*, Prentice Hall PTR publisher, 2001.

[6]. Kim Topley, *J2Me in a Nutshell*, O'Reilly publisher, 2002.

[7]. Gwenaël Le Bodic, *Mobile Messaging Technologies and Service*, John Wiley & Sons publisher, 2003.