TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

LẬP TRÌNH TRÊN DI ĐỘNG VỚI J2ME

GVHD: VÕ TÂM VÂN
SVTH : TRẦN ĐỨC MINH 0112355
VŨ THỌ TUẤN 0112411

04/2005

Mục lục

Trang

I. Giới thiệu về J2ME

1. Lịch sử

J2ME được phát triển từ kiến trúc Java Card, Embeded Java và Personal Java của phiên bản Java 1.1. Đến sự ra đời của Java 2 thì Sun quyết định thay thế Personal Java và được gọi với tên mới là Java 2 Micro Edition, hay viết tắt là J2ME. Đúng với tên gọi, J2ME là nền tảng cho các thiết bị có tính chất nhỏ, gọn:

	"Version 1"	Version 2
PCs	JDK™ 1.1.x	Java™ 2 Platform, Standard Edition (J2SE™)
PDAs and Communicators	PersonalJava™	Java™ 2 Platform, Micro Edition (J2ME™)/CDC
Phones and Pagers		J2ME/CLDC
Embedded Devices	EmbeddedJava™	J2ME/CLDC
Smart Cards	JavaCard™	JavaCard

2. Lý do chọn J2ME

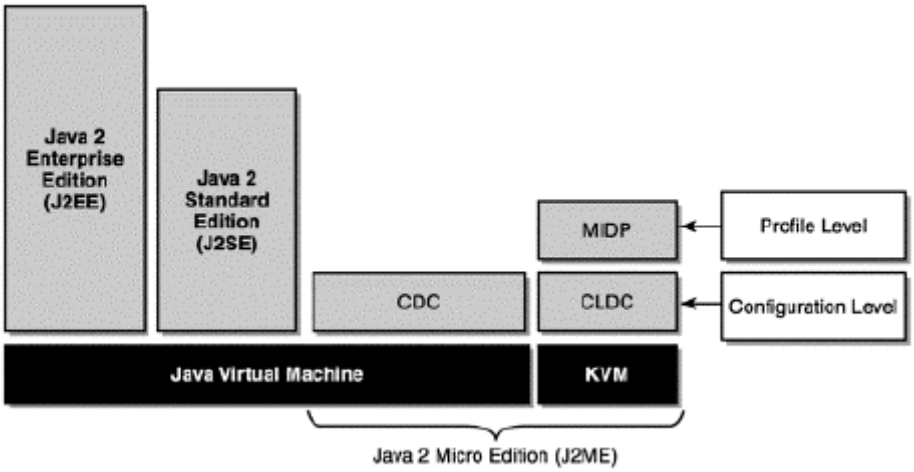
a) Java ban đầu được thiết kế dành cho các máy với tài nguyên bộ nhớ hạn chế.

b) 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 …

3. 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 tả định nghĩa một môi trường phần mềm cho một dòng các thiết bị được phân loại bởi tập hợp các đặc tính, ví dụ như:

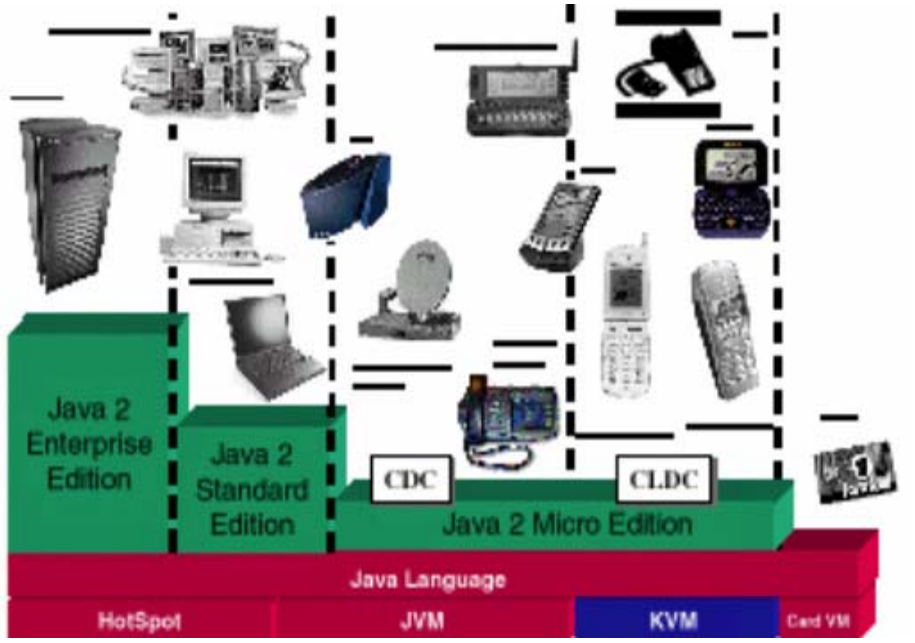
- Kiểu và số lượng bộ nhớ
- Kiểu và tốc độ bộ vi xử lý
- Kiểu mạng kết nối

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 bị cao 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 thoại 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 kế để nhắm vào thị trường các thiết bị cấp thấp (low-end), các thiết bị này thông thường là máy điện thoại 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 kể đến 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 hỗ 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

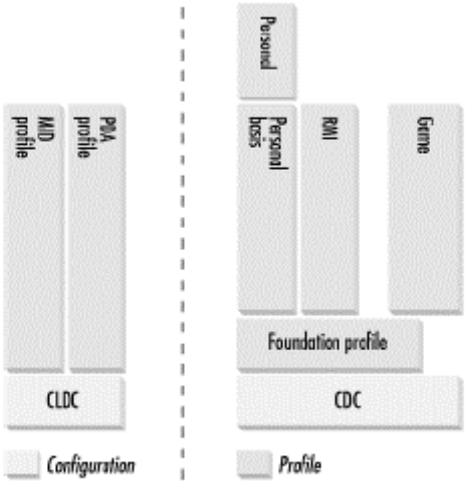
	CLDC	CDC
Ram	>=32K, <=512K	>=256K
Rom	>=128k, <=512k	>=512k
Nguồn Năng Lượng	Có GiớI Hạn (nguồn pin)	Không giới hạn
Network	Chậm	Nhanh

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 hỗ trợ J2ME. Sau đây là các profile tiêu biểu:

- Mobile Information Device Profile (MIDP): profile này sẽ bổ sung các tính năng như hỗ trợ 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 bộ nhớ 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.



4. Giới thiệu MIDP

a) Định nghĩa:

Đây là Profile được định nghĩa dành riêng cho các thiết bị di động và là thành phần chính trong J2ME. MIDP cung cấp các chức năng cơ bản cho hầu hết các dòng thiết bị di động phổ biến nhất như các máy điện thoại di động và các máy PDA. Tuy nhiên MIDP không phải là cây đũa thần cho mọi lập trình viên vì như chúng ta đã biết, MIDP được thiết kế cho các máy di động có cấu hình rất thấp. Trong phần sau tôi sẽ liệt kê qua các tính năng mà MIDP cung cấp và những giới hạn của nó.

b) Những chức năng MIDP không thực hiện được:

- Phép tính dấu phẩy động (floating point): Phép tính này đòi hỏi rất nhiều tài nguyên CPU và phần lớn các CPU cho các thiết bị di động không hỗ trợ phép tính này, do đó MIDP cũng không có.
- Bộ nạp class (Class Loader).
- Hỗ trợ từ khóa finalize() như trong J2SE: Việc “dọn dẹp“ tài nguyên trước khi nó bị xóa được đẩy về phía các lập trình viên.
- Không hỗ trợ JNI
- Hỗ trợ hạn chế thao tác bắt lỗi.
- 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.

c) Những chức năng MIDP cung cấp

- Các lớp và kiểu dữ liệu: Phần lớn các lớp mà các lập trình viên Java quen thuộc vẫn còn được giữ lại ví dụ như các lớp trong gói java.util như Stack, Vector và Hastable cũng như Enumeration.
- Hỗ trợ đối tượng Display: Đúng như tên gọi một chương trình MIDP sẽ hỗ trợ duy nhất một đối tượng Display là đối tượng quản lý việc hiển thị dữ liệu trên màn hình điện thoại.
- Hỗ trợ Form và các giao diện người dùng.
- Hỗ trợ Timer và Alert
- Cung cấp tính năng Record Management System (RMS) cho việc lưu trữ dữ liệu

Ngoài ra vào tháng 11 năm 2003 Sun đã tung ra MIDP 2.0 với hàng loạt tính năng khác được cung cấp thêm so với bản 1.0. Những cải tiến nổi bật so với MIDP 1.0

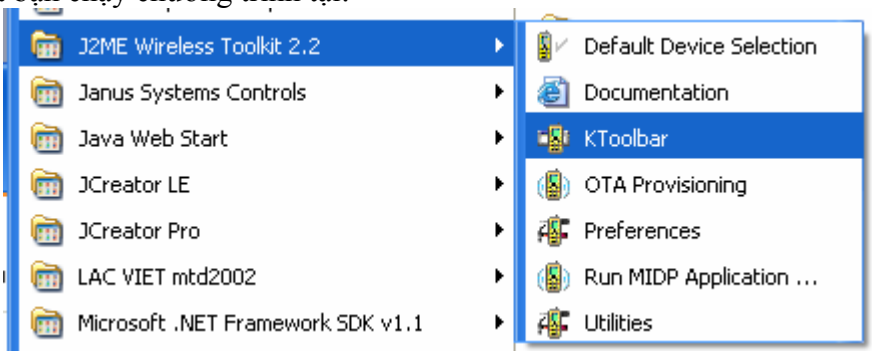
- Nâng cấp các tính năng bảo mật như
  - Download qua mạng an toàn hơn qua việc hỗ trợ giao thức HTTPS.
  - Kiểm soát việc kết nối giữa máy di động và server: ví dụ như các chương trình không thể kết nối tới server nếu thiếu sự chấp thuận của người sử dụng.
- Thêm các API hỗ trợ Multimedia. Một trong nhưng cải tiến hấp dẫn nhất của MIDP 2.0 là tập các API media của nó. Các API này là một tập con chỉ hỗ trợ âm thanh của Mobile Media API (MMAPI).
- Mở rộng các tính năng của Form. Nhiều cải tiến đã được đưa vào API javax.microedition.lcdui trong MIDP 2.0, nhưng các thay đổi lớn nhất (ngoài API cho game) là trong Form và Item.
- Hỗ trợ các lập trình viên Game bằng cách tung ra Game API: Có lẽ Sun đã kịp nhận ra thị trường đầy tiềm năng của các thiết bị di động trong lĩnh vực Game. Với MIDP 1.0 thì các lập trình viên phải tự mình viết code để quản lý các hành động của nhân vật cũng như quản lý đồ họa. Việc này sẽ làm tăng kích thước file của sản phẩm cũng như việc xuất hiện các đoạn mã bị lỗi. Được hưởng lợi nhất từ Game API trong MIDP 2.0 không chỉ là các lập trình viên Game mà còn là các lập trình viên cần sử dụng các tính năng đồ họa cao cấp. Ý tưởng cơ bản của Game API là việc giả định rằng một màn hình game là tập hợp các layer (lớp). Ví dụ như: trong một game đua xe thì màn hình nền là một layer, con đường là một layer và chiếc xe được xem như đang nằm trên layer khác. Với Game API nhà phát triển còn được cung cấp các tính năng như quản lý các thao tác bàn phím.
- Hỗ trợ kiểu ảnh RGB: một trong những cải tiến hấp dẫn cho các nhà phát triển MIDP là việc biểu diễn hình ảnh dưới dạng các mảng số nguyên, cho phép MIDlet thao tác với dữ liệu hình ảnh một cách trực tiếp.

5. Môi trường phát triển J2ME

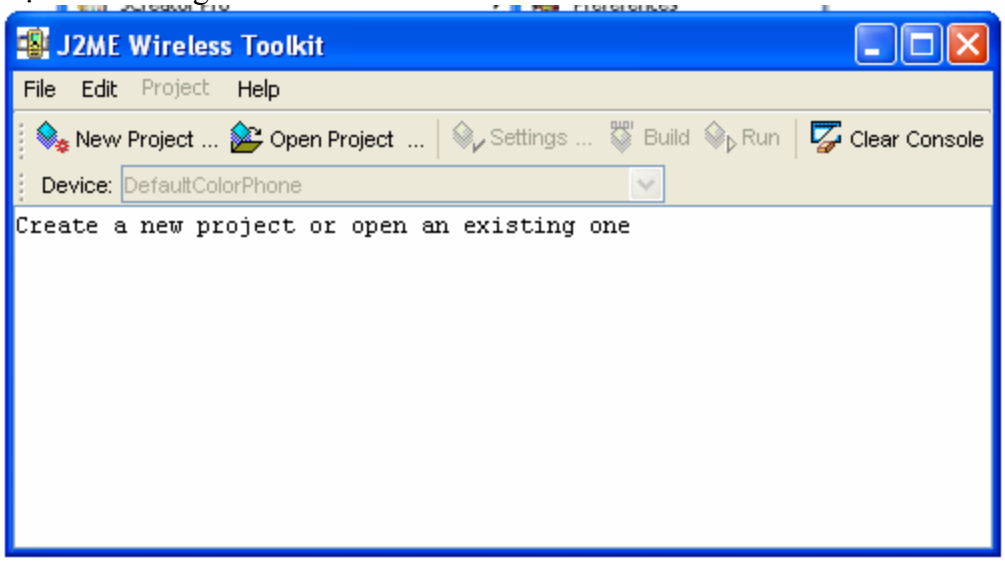
Có nhiều môi trường phát triển ứng dụng J2ME và mỗi hãng điện thoại cũng đưa ra những môi trường phát triển cho riêng mình. Ở đây tôi sẽ giới thiệu toolkit của Sun: J2ME Wireless Toolkit.

Địa chỉ download chương trình: http://java.sun.com/j2me/download.html. Ngoài ra bạn cần phải có J2SDK 1.4 trở lên.

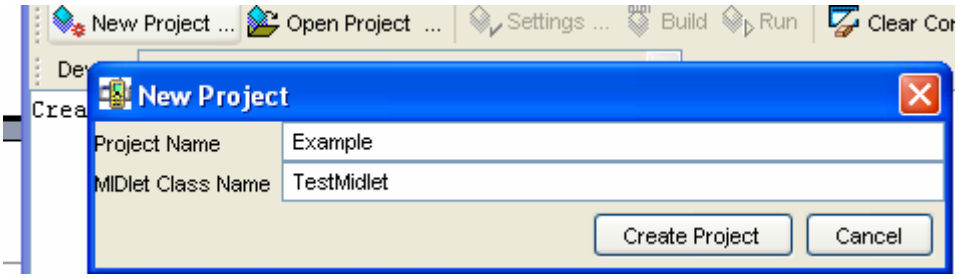
Sau khi cài đặt bạn chạy chương trình tại:



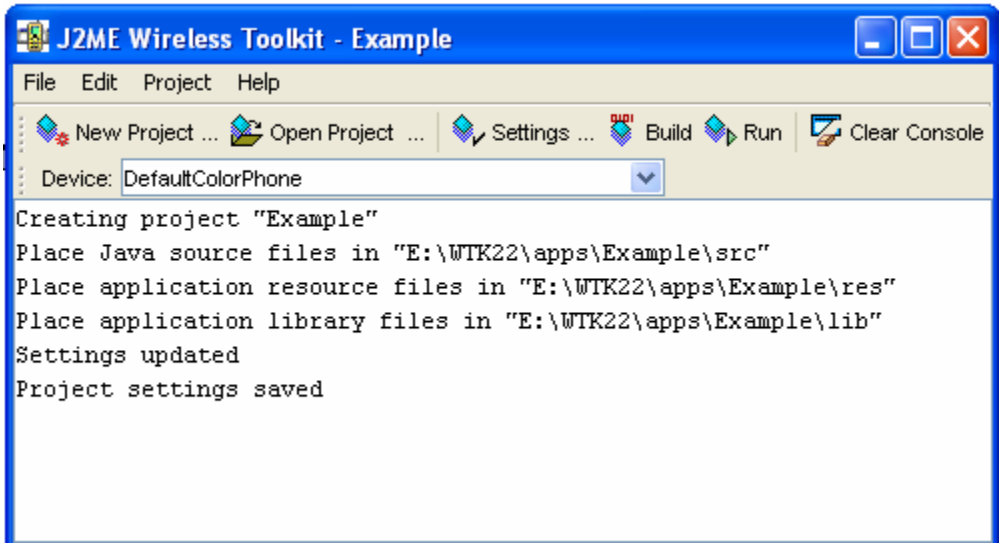
Giao diện của chương trình:



Sau khi hoàn thành các các bước ở trên, đây là lúc bắt đầu viết những dòng code đầu tiên. Bạn nhấp chuột vào nút bấm New Project và nhập vào tên Project và Class mà bạn muốn tạo, ví dụ TestMidlet.



Chú ý là bạn không nhất thiết phải tạo trùng tên class với tên Project. Trên cửa sổ chương trình sẽ hiển thị các thông báo về việc tạo các thư mục phục vụ cho việc xây dựng và thực thi mã chương trình

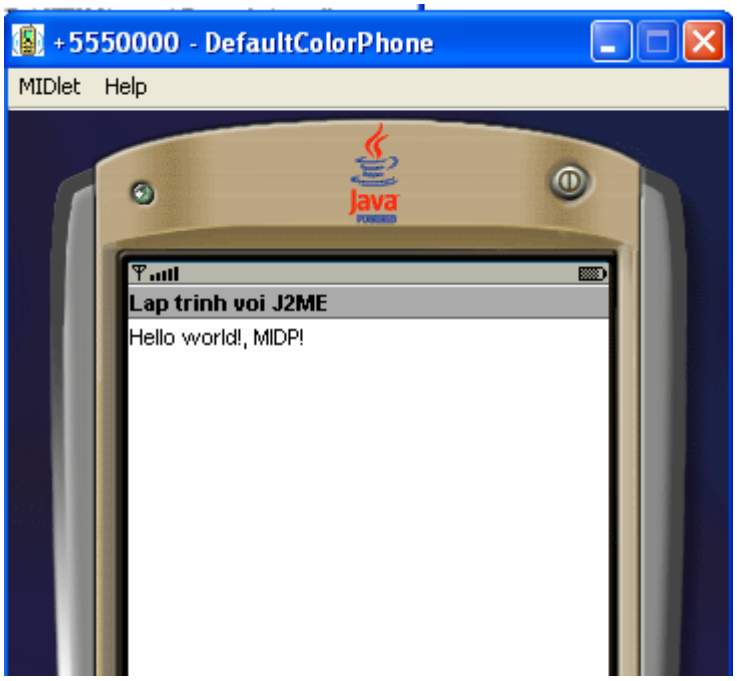


Hãy để ý thư mục : “E:\WTK22\apps\Example\src”, đây sẽ là nơi chứa source của ứng dụng.

Bạn có thể dùng bất kỳ chương trình soạn thảo văn bản nào để soạn code:

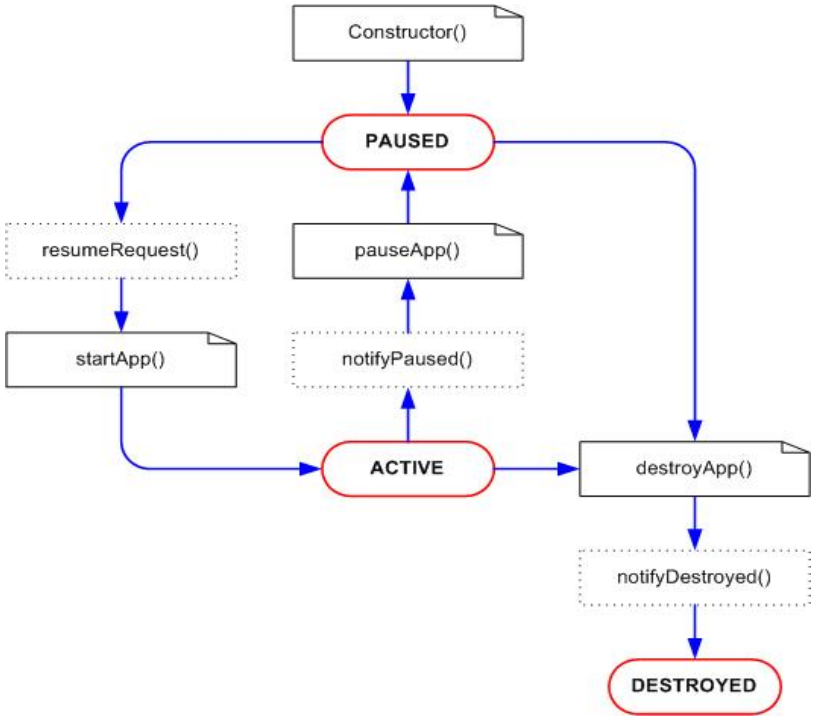
```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class TestMidlet
    extends MIDlet
    implements CommandListener {

  private Form mMainForm;
  public TestMidlet() {
    mMainForm = new Form("Lap trinh voi J2ME");
    mMainForm.append(new StringItem(null, "Hello world!, MIDP!"));
    mMainForm.addCommand(new Command("Exit", Command.EXIT, 0));
    mMainForm.setCommandListener(this);
  }
  public void startApp() {
    Display.getDisplay(this).setCurrent(mMainForm);
  }

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

Sau đó copy source này vào thư muc nói trên. Rồi tiến hành build và run chương trình



Vòng đời của một MIDlet

Giống như dạng chương trình Applet trên J2SE, một Midlet luôn luôn kế thừa javax.microedition.midlet Hàm cơ bản nhất trong mọi Midlet là startApp(), hàm này sẽ khởi tạo Midlet cũng như vận hành các thành phần hoặc đối tượng khác, ngoài startApp(), mỗi Midlet còn có pauseApp() và destroyApp(), mỗi hàm này sẽ được gọi thực thi tương ứng khi user chọn *dừng* hoặc *thoát* chương trình.



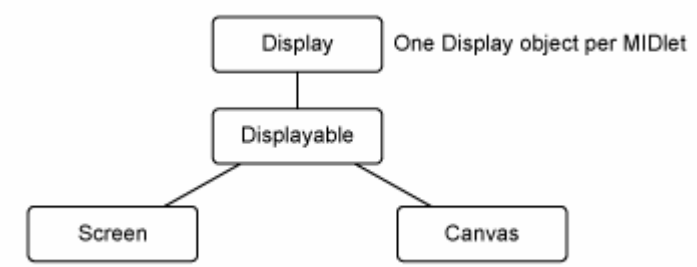
II. Các thành phần giao diện ở mức cao của ứng dụng MIDP

1. Đối tượng Display, Displayable và Screens

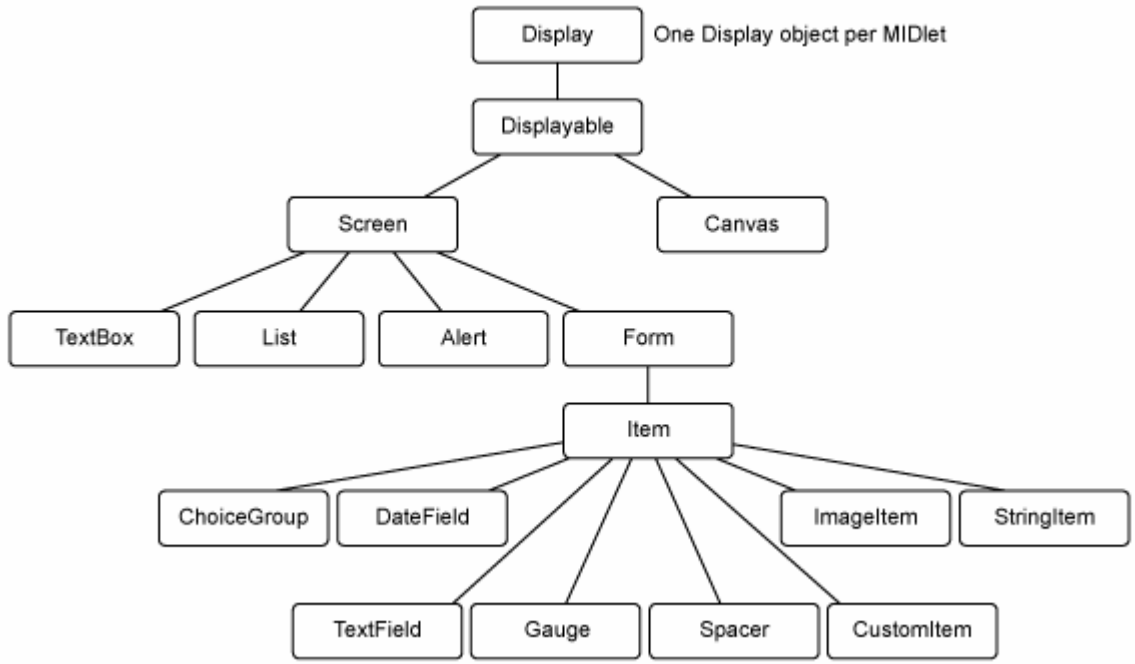
Một ứng dụng MIDlet chỉ có 1 đối tượng thể hiện **Display**. Đối tượng này dùng để lấy thông tin về đối tượng trình bày, ví dụ màu được hỗ trợ, và bao gồm các phương thức để yêu cầu các đối tượng được trình bày. Đối tượng **Display** cần thiết cho bộ quản lý việc trình bày trên thiết bị điều khiển thành phần nào sẽ được hiển thị lên trên thiết bị

Mặc dù chỉ có một đối tượng **Display** ứng với mỗi MIDlet, nhưng nhiều đối tượng trong một MIDlet có thể được hiển thị ra trên thiết bị như **Forms**, **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 sẽ được thừa kế bởi các thành phần hiển thị ở mức cao, chính các thành phần này sẽ được 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. 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 trên:

- 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 một đối tượng DateField với nhãn, cho phép 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());

    // All the commands/buttons
    cmExit = new Command("Exit", Command.EXIT, 1);

    // Add to form and listen for events
    fmMain.append(dfAlarm);
    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 độ hoàn 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 dữ liệu 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 nhâp vào các địa chỉ email hợp lệ
- **NUMERIC**: chỉ cho phép nhập số
- **PHONENUMBER**: Chỉ cho phép nhập số điện 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:

```
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(chỉ được 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. Bạn có thể dùng Spacer để chỉ rõ khoãng 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

b. 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 thể được 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)
    {
      ...
    }

  ...

  }
```

g) 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 sẽ được 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

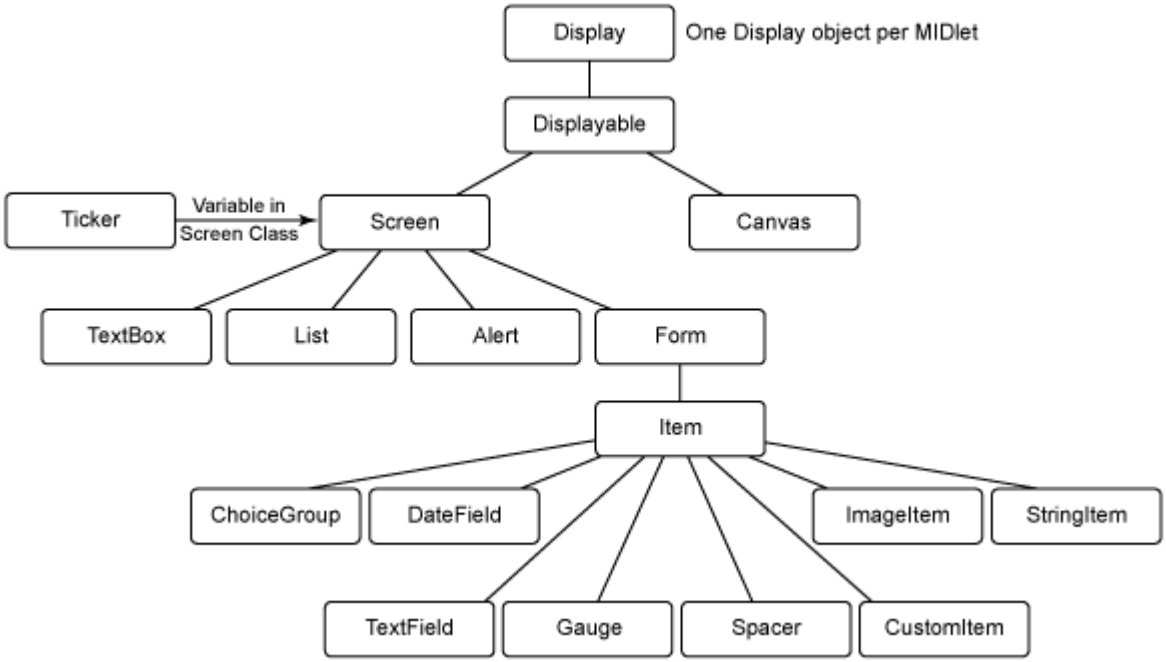
Đoạn mã dưới đây mô tả việc sử dụng đối tượng **Image** và đối tượng **ImageItem**

```
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();
    }
  }
}
```

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 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 chỉ định 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)

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

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

public class TextBoxTest extends MIDlet implements CommandListener
{
  private Display display;              // Reference to Display object
  private TextBox tbClip;       // Main textbox
  private Command cmExit;       // Command to exit

  public TextBoxTest()
  {
    display = Display.getDisplay(this);

    // Create the Commands. Notice the priorities assigned
    cmExit = new Command("Exit", Command.EXIT, 1);

    tbClip = new TextBox("Textbox Test", "Contents go here..",125, TextField.ANY);
    tbClip.addCommand(cmExit);
    tbClip.setCommandListener(this);

  }

  public void startApp()
  {
    display.setCurrent(tbClip);
  }

  public void pauseApp()
```

```
 {
 }

 public void destroyApp(boolean unconditional)
 {
 }

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

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 chỉ được 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
  cgSound = new ChoiceGroup("Choose a sound", Choice.EXCLUSIVE);

  // Append options, with no associated images
  cgSound.append("Info", null);
  cgSound.append("Confirmation", null);
  cgSound.append("Warning", null);
  cgSound.append("Alarm", null);
  cgSound.append("Error", null);

  cmExit = new Command("Exit", Command.EXIT, 1);

  // Create Form, add components, listen for events
  fmMain = new Form("");
  fmMain.append(cgSound);
  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 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", "Confirmation sound",
                        null, AlertType.INFO);
              break;

            case 2:
              al = new Alert("Alert sound", "Warning sound",
                        null, AlertType.INFO);
              break;

            case 3:
              al = new Alert("Alert sound", "Alarm sound",
                        null, AlertType.INFO);
              break;

            case 4:
              al = new Alert("Alert sound", "Error sound",
                        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();
    }
  }
}
```

III. Các thành phần giao diện ở mức thấp của ứng dụng MIDP

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. Bạn 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()**

2. Lớp Canvas

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 sẽ minh 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 độ để 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 vẽ độ 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, bạn 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 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 chỉ hướng của thiết bị, nhưng không phải tất cả các thiết bị di động đều có những giá trị này. Nếu một thiết bị di động thiếu các phím mũi tên thì các hành động của trò chơi sẽ được 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;
```

```
/*-------------------------------------------------
* Constructor
*-------------------------------------------------*/
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);
}

/*-------------------------------------------------
* Paint the text representing the key code
*-------------------------------------------------*/
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);
  }
}

/*-------------------------------------------------
* Command event handling
*-------------------------------------------------*/
public void commandAction(Command c, Displayable d)
{
  if (c == cmExit)
    midlet.exitMIDlet();
}

/*-------------------------------------------------
* Key code event handling
*-------------------------------------------------*/
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 sự kiện này được thiết kế để 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. Phương thức **hasPointerMotionEvents()** trả về một giá trị có kiểu **boolean** nhằm chỉ rõ răng 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;

  /*-------------------------------------------------
  * Constructor
  *-------------------------------------------------*/
  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);
  }

  /*-------------------------------------------------
  * Draw line based on start and ending points
  *-------------------------------------------------*/
  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;
  }

  /*-------------------------------------------------
  * Command event handling
  *-------------------------------------------------*/
  public void commandAction(Command c, Displayable d)
  {
    if (c == cmExit)
      midlet.exitMIDlet();
    else if (c == cmClear)
    {
      clearDisplay = true;
      repaint();
    }
  }

  /*-------------------------------------------------
  * Pointer pressed
  *-------------------------------------------------*/
  protected void pointerPressed(int x, int y)
  {
    startx = x;
    starty = y;
  }

  /*-------------------------------------------------
  * Pointer moved
  *-------------------------------------------------*/
  protected void pointerDragged(int x, int y)
  {
    currentx = x;
    currenty = y;
    repaint();
  }
}
```

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 sẽ được hiển thị ra trên thiết bị.

```
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 bạn có thể xác định màu bằng 2 cách. Cách 1: bạn có thể 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. Hay cách 2 là bạn có thể dùng từng tham số riêng biệt để xác định mỗi màu. Khi sử dụng một giá trị để 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 sẽ hướng dẫn bạn 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 bạn 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 vẽ đường 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);
hay
g.setStrokeStyle(Graphics.SOLID);
```

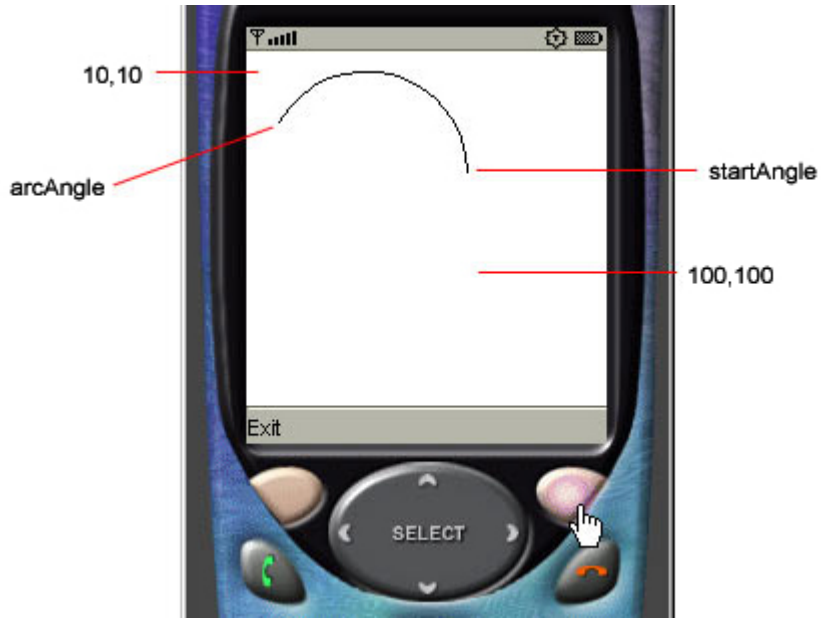
c) Vẽ cung

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 chỉ định chiều bao quanh bên ngoài của một hình hộp chữ nhật tưởng tượng. Góc bắt đầu xác định vị trí bắt đầu vẽ khung, 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 độ điểm 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
*
* Draw arcs
*-------------------------------------------------*/
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);
  }

  /*-------------------------------------------------
  * Draw shapes
  *-------------------------------------------------*/
  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 150 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 chỉ được 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 chữ được 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 hay. Ví dụ

```
Font font = Font.getFont(Font.FACE_PROPORTIONAL,
Font.STYLE_BOLD | Font.STYLE_ITALIC, Font.SIZE_MEDIUM);
```

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

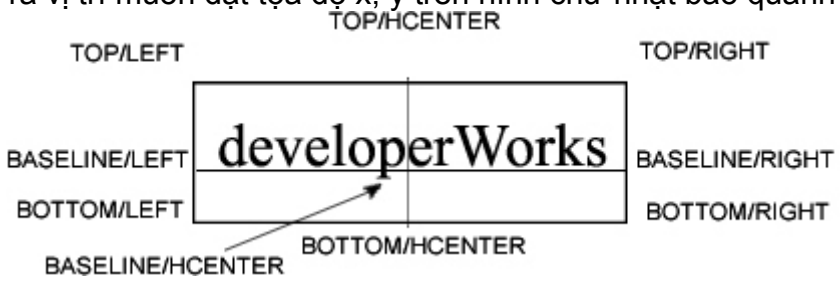
```
int getFace()
int getStyle()
int getSize()
boolean isPlain()
boolean isBold()
boolean isItalic()
boolean isUnderlined()
```

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 tương 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 tự được hiển thị, thì điểm neo cho phép bạn chỉ ra vị trí muốn đặt tọa độ x, y trên hình chữ nhật bao quanh chuỗi 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), bạn phải chọn một điểm hoành độ và một điểm tung độ. Các điểm neo được định nghĩa như ở dưới đây

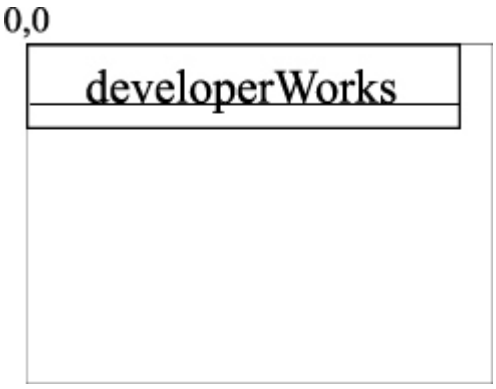
Chiều ngang
LEFT (Bên trái)
HCENTER (Chính giữa của chiều ngang)
RIGHT (Bên phải)

Chiều dọc
TOP (Ở trên)
BASELINE (Đường thẳng cơ sở)
BOTTOM (Ở dưới)

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

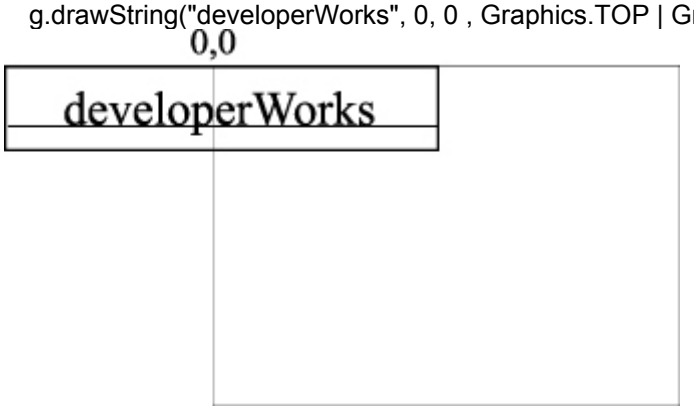
```
g.drawString("developerWorks", 0, 0 , Graphics.TOP | Graphics.LEFT);
```

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.drawString("developerWorks", 0, 0 , Graphics.TOP | Graphics.HCENTER);
```



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_ITALICS,
Font.SIZE_MEDIUM));

  // 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);
   notifyDestroyed();
  }
 }

 /*-------------------------------------------------
 * FontCanvas.java
```

```
*
* Draw text using specified Font
*-------------------------------------------------*/
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);
  }

  /*-------------------------------------------------
  * Draw text
  *-------------------------------------------------*/
  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);
  }
}

/*-------------------------------------------------
* PrefsForm.java
*
* Form that allows user to select font preferences
*-------------------------------------------------*/
import javax.microedition.lcdui.*;

public class PrefsForm extends Form implements
CommandListener, ItemStateListener
{
  private FontViewer midlet;      // Main midlet
  private Command cmBack,         // Back to canvas
                  cmSave;         // Save new font prefs
```

```
protected ChoiceGroup cgFace,   // Font face
            cgStyle,        // style
            cgSize;         // size
protected TextField tfText;     // Text string to display on canvas
                        // (with the requested font prefs)
          private int face = 0,          // Values for font attributes
          style = 0,          // that are obtained from the choicegroups
                    size = 0;
private String text = null;     // Value for text string from the textfield

public PrefsForm(String title, FontViewer midlet)
{
  // Call the form constructor
  super(title);

  // Save reference to MIDlet so we can
  // access the display manager class
  this.midlet = midlet;

  // Commands
  cmSave = new Command("Save", Command.SCREEN, 1);
  cmBack = new Command("Back", Command.BACK, 2);

    // Exclusive choice group for font face
  cgFace = new ChoiceGroup("Face Options", Choice.EXCLUSIVE);
  cgFace.append("System", null);
  cgFace.append("Monospace", null);
  cgFace.append("Proportional", null);

  // Multiple choice group for font style
  cgStyle = new ChoiceGroup("Style Options", Choice.MULTIPLE);
  cgStyle.append("Plain", null);
  cgStyle.append("Bold", null);
  cgStyle.append("Italic", null);
  cgStyle.append("Underlined", null);

  // Exclusive choice group for font size
  cgSize = new ChoiceGroup("Size Options", Choice.EXCLUSIVE);
  cgSize.append("Small", null);
  cgSize.append("Medium", null);
  cgSize.append("Large", null);

  // Textfield for text to display on canvas
  tfText = new TextField("", "developerWorks", 20, TextField.ANY);

  // Create form contents and listen for events
  append(cgFace);
  append(cgStyle);
  append(cgSize);
  append(tfText);
  addCommand(cmSave);
```

```
  addCommand(cmBack);
  setCommandListener(this);
  setItemStateListener(this);
}

/*-----------------------------------------------
* Command event handling
*-----------------------------------------------*/
public void commandAction(Command c, Displayable s)
{
  if (c == cmSave)
  {
    // Set the values in canvas class to those selected here
    midlet.cvFont.setFace(face);
    midlet.cvFont.setStyle(style);
    midlet.cvFont.setSize(size);

    // Make sure we aren't passing a null value
    midlet.cvFont.setText(tfText.getString() !=
    null ? tfText.getString() : "developerWorks");
  }

  // No specific check needed for "Back" command
  // Always return to the canvas at this point
  midlet.showCanvas();
}

public void itemStateChanged(Item item)
{
  if (item == cgFace)
  {
            switch (cgFace.getSelectedIndex())
            {
      case 0:
        face = Font.FACE_SYSTEM;
        break;
      case 1:
        face = Font.FACE_MONOSPACE;
        break;
      case 2:
        face = Font.FACE_PROPORTIONAL;
        break;
    }
  }
  else if (item == cgStyle)
  {
    boolean[] b = new boolean[4];
    cgStyle.getSelectedFlags(b);

    style = 0;
```

```
        // STYLE_PLAIN has a value of 0
        //   No need to check for b[0]

        // If bold selected
                if (b[1])
          style = Font.STYLE_BOLD;

        // If italic selected
                if (b[2])
          style |= Font.STYLE_ITALIC;

        // If underlined selected
                if (b[3])
          style |= Font.STYLE_UNDERLINED;
      }
      else if (item == cgSize)
      {
                switch (cgSize.getSelectedIndex())
                {
          case 0:
            size = Font.SIZE_SMALL;
            break;
          case 1:
            size = Font.SIZE_MEDIUM;
            break;
          case 2:
            size = Font.SIZE_LARGE;
            break;
        }
      }
      else if (item == tfText)
      {
        text = tfText.getString();
      }
    }
  }
```

h) Vẽ ảnh

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

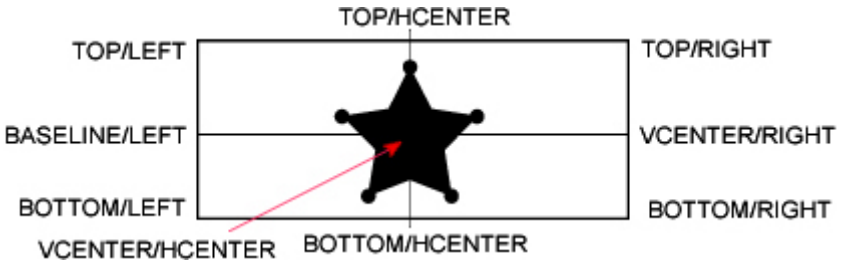
drawImage(Image img, int x, int y, int anchor)

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 như điểm 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 thế **VCENTER** 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 từ những đống tạp nham, chúng ta sẽ 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 đỏi 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
*
* Draw mutable image
*-------------------------------------------------*/
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");
    }
  }

  /*-------------------------------------------------
  * Draw mutable image
  *-------------------------------------------------*/
  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()
```

4. Các hàm API dùng để lập trình Game

Các hàm API dành để lập trình Game được giới thiệu trong bản MIDP 2.0, những hàm này là phương tiện để phát triển game với nhiều phần đồ họa. Các hàm API dành cho Game là một phần của gói **javax.microedition.lcdui.game**

Lớp GameCanvas gần như giống lớp Canvas, GameCanvas cung cấp nền tảng để tạo giao diện người dùng, nhưng trong trường hợp này chỉ cho việc tạo games. GameCanvas chứa một vùng nhớ tách rời với vùng nhớ màn hình cho mỗi thể hiện và cung cấp các phương tiện tích hợp để xác định tình trạng các phím trò chơi. Dưới đây là lớp GameCanvas:

public abstract class **GameCanvas** extends **Canvas**

Layer là một lớp trừu tượng được sử dụng để thể hiện một đối tượng trực quan trong một trò chơi. Sprite là một lớp con của lớp Layer, lớp này được cung cấp để thể hiện cho một bức ảnh. Ngoài ra Sprite còn có thể bao gồm một dãy các khung ảnh. Để thực hiện khả năng chuyển động, các khung ảnh được thể hiện theo một thứ tự để tạo hiệu ứng di chuyển ảnh. Các phép biến đổi như là phép quay và phép lật ảnh có thể được áp dụng đối với một đối tượng Sprite. Dưới đây là mô tả cho 2 lớp Layer và Sprite

public abstract class **Layer** extends Object

public class **Sprite** extends **Layer**

TiledLayer là một lớp tương tự như một bảng tính, với mỗi ô đại diện cho một tấm ảnh. Một TiledLayer đơn giản được dùng để thể hiện các phần tử trực quan có kích thước lớn, như là nền của một trò chơi.

public class **TiledLayer** extends Layer

Để đơn giản xử lý cho việc vẽ nhiều lớp trong một trò chơi, các hàm API dành cho việc lập trình trò chơi lại có thê lớp LayoutManager. Lớp LayoutManager này chứa một danh sách có thứ tự các đối tượng Layers và xác định khu vực nào cần được vẽ lại và thể hiện theo đúng trật tự. Lớp LayoutManager được thể hiện như dưới đây:

public class LayerManager extends Object

IV. Xử lý sự kiện

1. Đối tượng Command

Khi một đối tượng xảy ra trên thiết bị di động, một đối tượng **Command** giữ thông tin về sự kiện đó. Thông tin này bao gồm loại hành động thực thi, nhãn của mệnh lệnh và độ ưu tiên của chính nó. Trong J2ME, các hành động nói chung được thể hiện dưới dạng các nút trên thiết bị

Nếu có quá nhiều hành động được hiển thị trên thiết bị, thiết bị sẽ tạo ra một thực đơn để chứa các hành động

Chỉ có các thành phần MIDP sau đây mới có thể chứa các đối tượng **Command** là: **Form**, **TextBox**, **List** và **Canvas**.

Các bước cơ bản để xử lý các sự kiện với một đối tượng **Command**:

- Tạo một đối tượng **Command**
- Đặt đối tượng Command lên trên một đối tượng Form, TextBox, List, hay Canvas
- Tạo một bộ lắng nghe

Khi có một sự kiện xảy ra, bộ lắng nghe sẽ phát sinh một lời gọi đến phương thức **commandAction()**. Trong thân phương thức này bạn có thể xác định đối tượng nào phát sinh ra sự kiện và tạo ra các xử lý tương ứng

Dưới đây là đoạn mã minh họa việc tạo ra sự kiện Command và xử lý sự kiện:

```
private Form fmMain;              // Form
private Command cmExit;           // Command to exit the MIDlet
        ...
fmMain = new Form("Core J2ME");   // Create Form and give it a title

// Create Command object, with label, type and priority
cmExit = new Command("Exit", Command.EXIT, 1);
...
fmMain.addCommand(cmExit);        // Add Command to Form
fmMain.setCommandListener(this);  // Listen for Form events

...

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

2. Đối tượng Item

Ngoài việc xử lý sự kiện bằng đối tượng **Command** ta có thể xử lý sự kiện bằng đối tượng **Item**. Nhiều đối tượng trong MIDP đã được định nghĩa trước cho việc xử lý các sự kiện cụ thể nào đó. Ví dụ đối tượng DateField cho phép người sử dụng chọn lựa ngày và giờ trên màn hình, đối tượng TextField cho phép người dùng nhập vào một chuỗi các ký tự, số và các ký tự đặc biệt

Tương tự như việc xử lý sự kiện bằng đối tượng **Command**, thì khi có một thay đổi xảy ra đối với bất kỳ thành phần **Item** nào thì phương thức **itemStateChanged()** sẽ được gọi. Và chúng ta sẽ thực hiện các xử lý bên trong phương thức này

Dưới đây là đoạn mã minh họa việc tạo ra sự kiện Command và xử lý sự kiện:

```
private Form fmMain;                // Form
private DateField dfToday;          // DateField item
...
fmMain = new Form("Core J2ME");                  // Create Form object
dfToday = new DateField("Today:", DateField.DATE);  // Create DateField
...
fmMain.append(dfToday);             // Add DateField to Form
fmMain.setItemStateListener(this);  // Listen for Form events

...

public void itemStateChanged(Item item)
{
  // If the datefield initiated this event
  if (item == dfToday)
    ...
}
```

3. Ví dụ

Dưới đây là đoạn mã minh họa việc tạo ra và xử lý các sự kiện Command, Item

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

public class EventProcessing extends MIDlet implements
ItemStateListener, CommandListener
{
    private Display display;    // Reference to Display object for this MIDlet
    private Form fmMain;        // Main Form
    private Command cmExit;     // Command to exit the MIDlet
    private TextField tfText;   // TextField component (item)
```

```
    public EventProcessing()
    {
      display = Display.getDisplay(this);

      // Create the date and populate with current date
      tfText = new TextField("First Name:", "", 10, TextField.ANY);

      cmExit = new Command("Exit", Command.EXIT, 1);

      // Create the Form, add Command and DateField
      // listen for events from Command and DateField
      fmMain = new Form("Event Processing Example");
      fmMain.addCommand(cmExit);
      fmMain.append(tfText);
      fmMain.setCommandListener(this);     // Capture Command events (cmExit)
      fmMain.setItemStateListener(this);   // Capture Item events (dfDate)
    }

    // 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)
    {
      System.out.println("Inside commandAction()");

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

    public void itemStateChanged(Item item)
    {
      System.out.println("Inside itemStateChanged()");
    }
}
```

V. Record Management System

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. Persistent Storage Through the Record Store

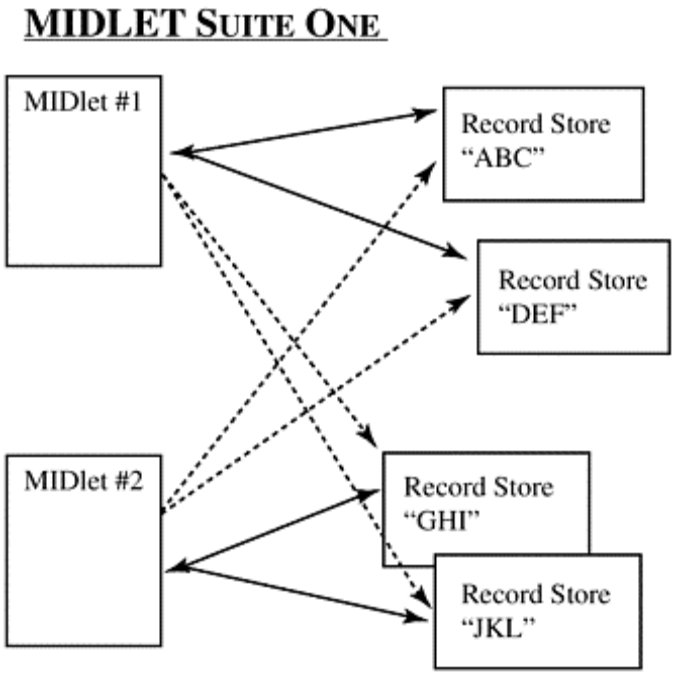
RMS là hệ thống được tổ chức và quản lý dưới dạng các record (bản ghi). Mỗi bản ghi (sau này gọi là Record) có thể chứa bất kỳ loại dữ liệu nào, chúng có thể là kiểu số nguyên, chuỗi ký tự hay có thể là một ảnh và kết quả là một Record là một chuỗi (mảng) các byte. Nếu bạn mã hoá dữ liệu của bạn dưới 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:

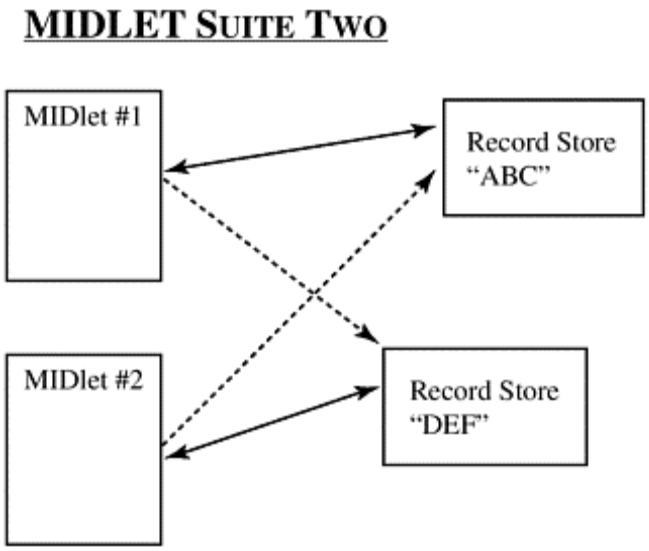
A record store database

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 thể đứng 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 số định danh kiểu số nguyên gọi là Record ID. Record đầu tiên được tạo ra sẽ được 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 dung 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 sẽ được 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 bạn 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, bạn 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 bạn 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 bạn 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 bạn. Vì vậy trong kỹ thuật lập trình bạn 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 bạn chỉ được sử dụng bởi một MIDlet, bạn 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 2 ví dụ đơn giản của việc đọc ghi record trong RecordStore.

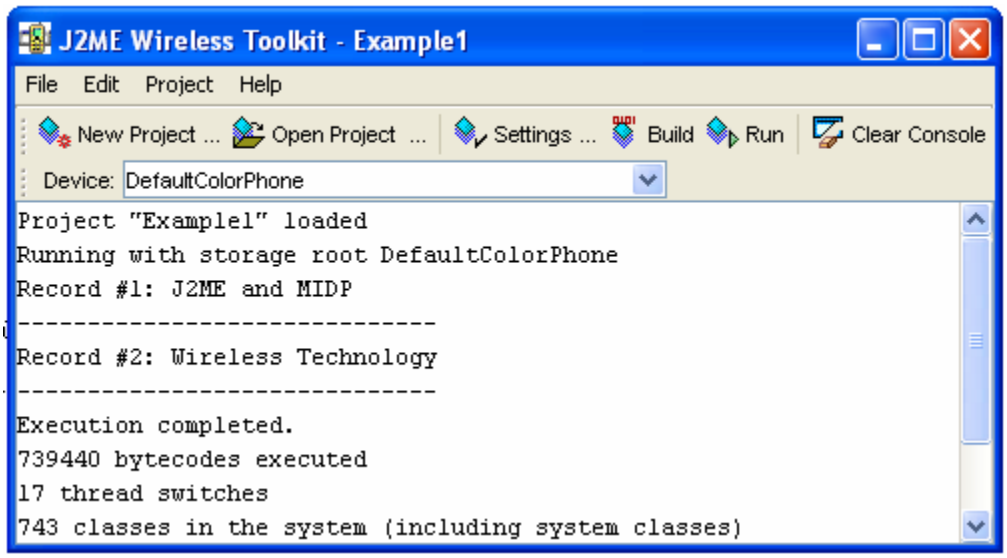
Ví dụ 1: đọc và ghi đối tượng string (ReadWrite.java)

```
/*-------------------------------------------------
* ReadWrite.java
*
* Read and write to the record store.
*
* No GUI interface, all output is to the console
*-------------------------------------------------*/
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());
		}
	}
		/*-------------------------------------------------
		* Simple message to console for debug/errors
		* When used with Exceptions we should handle the
		* error in a more appropriate manner.
		*-------------------------------------------------*/
	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 ví dụ 2 này, chương trình sẽ 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.

Ví dụ 2: đọc và ghi sử dụng stream (ReadWriteStreams.java)

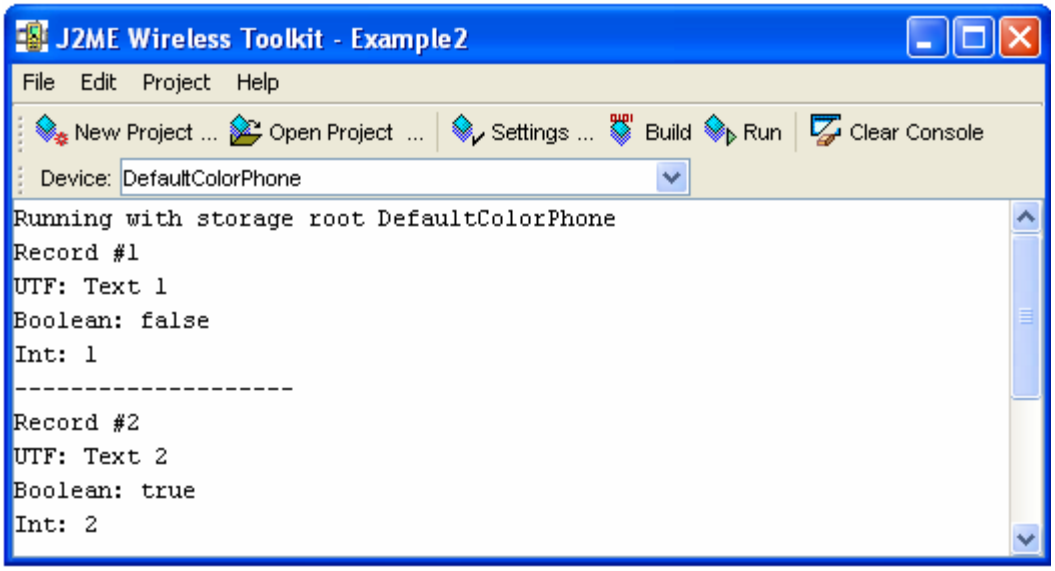
```
/*-------------------------------------------------
* ReadWriteStreams.java
*
* Use streams to read and write Java data types
* to the record store.
*
* No GUI interface, all output is to the console
*-------------------------------------------------*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
public class ReadWriteStreams extends MIDlet
{
        private RecordStore rs = null; // Record store
        static final String REC_STORE = "db_1"; // Name of record store
        public ReadWriteStreams()
        {
                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());
		}
	}
}
/*-------------------------------------------------
* Create three arrays to write to record store
*-------------------------------------------------*/
public void writeTestData()
{
	String[] strings = {"Text 1", "Text 2"};
	boolean[] booleans = {false, true};
	int[] integers = {1 , 2};
	writeStream(strings, booleans, integers);
}
/*-------------------------------------------------
* Write to record store using streams.
*-------------------------------------------------*/
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());
        }
}
/*-------------------------------------------------
* Read from the record store using streams
*-------------------------------------------------*/
public void readStream()
{
        try
        {
                // Careful: Make sure this is big enough!
                // Better yet, test and reallocate if necessary
                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);
                for (int i = 1; i <= rs.getNumRecords(); i++)
                {
```

```
                        // Get data into the byte array
                        rs.getRecord(i, recData, 0);
                        // Read back the data types
                        System.out.println("Record #" + i);
                        System.out.println("UTF: " + strmDataType.readUTF());
                        System.out.println("Boolean: " +
                        strmDataType.readBoolean());
                        System.out.println("Int: " + strmDataType.readInt());
                        System.out.println("--------------------");
                        // Reset so read starts at beginning of array
                        strmBytes.reset();
                }
                strmBytes.close();
                strmDataType.close();
        }
        catch (Exception e)
        {
                db(e.toString());
        }
}
/*-------------------------------------------------
* Simple message to console for debug/errors
* When used with Exceptions we should handle the
* error in a more appropriate manner.
*-------------------------------------------------*/
private void db(String str)
{
        System.err.println("Msg: " + str);
}
}
```

Output của ví dụ 2:



Dữ liệu ghi vào record trong ví dụ 2 gồm:

```
String[] strings = {"Text 1", "Text 2"};
boolean[] booleans = {false, true};
int[] integers = {1 , 2};
```

Hàm WriteStream sẽ ghi dữ liệu vào RecordStore

```
// 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();
}
```

Hàm đã sử dụng hai stream là strmBytes là một mảng byte sẽ ghi vào RecordSotre và strmDataType để chứa dữ liệu. Như vậy quá trình ghi dữ liệu vào RecordStore được thực hiện thông qua các bước:

- Cấp phát stream.
- Ghi dữ liệu vào stream.
- Flush stream.
- Chuyển đổi stream data thành mảng byte.
- Ghi mảng byte vào RecordStore.
- Đóng stream.

Việc đọc các dữ liệu từ record đơn giản chỉ cần thay OutStream thành InputStream:

```
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);
for (int i = 1; i <= rs.getNumRecords(); i++)
{
        // Get data into the byte array
        rs.getRecord(i, recData, 0);
        // Read back the data types
        System.out.println("Record #" + i);
```

```
            System.out.println("UTF: " + strmDataType.readUTF());
            System.out.println("Boolean: " + strmDataType.readBoolean());
            System.out.println("Int: " + strmDataType.readInt());
            System.out.println("--------------------");
            // Reset so read starts at beginning of array
            strmBytes.reset();
    }
```

Ở đây, chúng tôi khai báo một mảng byte recData (giống với strmBytes), sau khi gọi hàm

```
rs.getRecord(i, recData, 0);
```

Chúng ta đã có mảng byte lúc đầu chương trình ghi vào record, sau đó dùng InputStream strmDataType để đọc đúng kiểu dữ liệu:

```
System.out.println("UTF: " + strmDataType.readUTF());
System.out.println("Boolean: " + strmDataType.readBoolean());
System.out.println("Int: " + strmDataType.readInt());
```

Lưu ý: khi sử dụng DataOutputStream và DataInputStream, cần phải đọc và ghi theo đúng thứ tự, nếu không sẽ không ra kết quả mong muốn.

4. Duyệt Record với RecordEnumeration

Trong các ví dụ trước để duyệt các record ta đã sử dụng vòng lặp:

```
for (int i = 1; i <= rs.getNumRecords(); i++)
{
        rs.getRecord(i, recData, 0);
...
}
```

Ngoài ra còn một cách khác để duyệt RecordStore là sử dung **RecordEnumeration**. Lớp này cung cấp các phương thức để duyệt các record trong RecordStore một cách nhanh chóng. Dưới đây là đoạn code duyệt toàn bộ RecordStore:

```
RecordEnumeration re = rs.enumerateRecords(null,null,false);
while (re.hasNextElement())
{
        // Get the next record into a String
        String str = new String(re.nextRecord());
        ... do something ...
}
```

Trong đoạn code trên đã 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 sự thay đổi
boolean isKeptUpdated()	Kiểm tra enumeration có tự động 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

5. 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 3: chương trình sắp xếp cơ bản

```
/*--------------------------------------------------
* SimpleSort.java
*
* No GUI interface, all output is to the console
*--------------------------------------------------*/
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());
	}
}
```

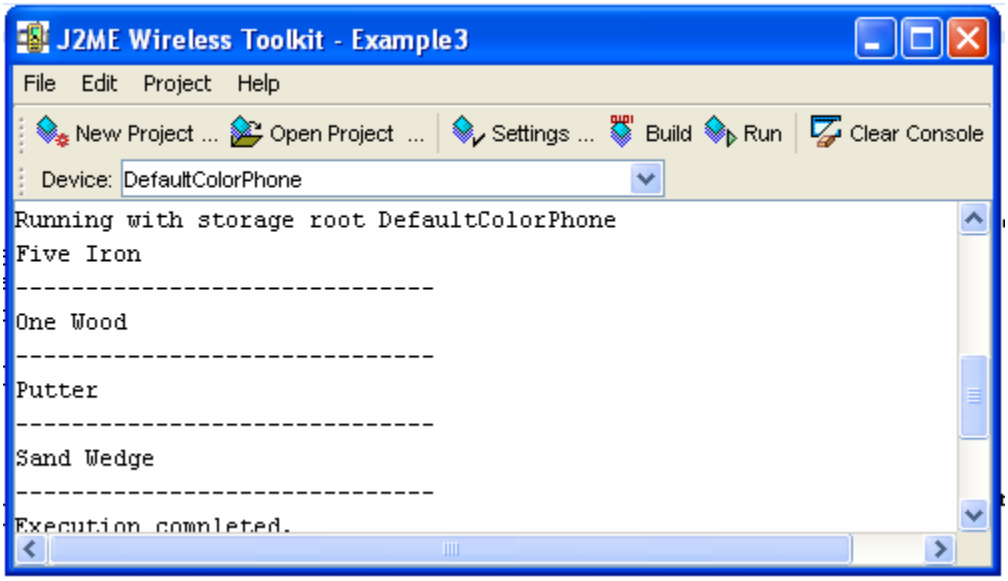
```
	public void readRecords()
	{
		try
		{
			if (rs.getNumRecords() > 0)
			{
				Comparator comp = new Comparator();
				RecordEnumeration re = rs.enumerateRecords(null,comp, false);
				while (re.hasNextElement())
				{
					String str = new String(re.nextRecord());
					System.out.println(str);
					System.out.println("------------------------------");
				}
			}
		}
		catch (Exception e)
		{
			db(e.toString());
		}
	}
	/*-------------------------------------------------
	* Simple message to console for debug/errors
	* When used with Exceptions we should handle the
	* error in a more appropriate manner.
	*-------------------------------------------------*/
	private void db(String str)
	{
		System.err.println("Msg: " + str);
	}
}
	/*-------------------------------------------------
	| Comparator.java
	|
	| Compares two records to determine sort order
	*-------------------------------------------------*/
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 3:



Ví dụ trên đúng trong trường hợp dữ liệu lưu vào record là dạng text, nếu quay lại ví dụ 2 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);
```

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 3 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 để sắp xếp các record, và việc lựa chọn này tùy thuộc vào ứng dụng của bạn.

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 3 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, bạn cần phải lấy ra từ mỗi record trường dữ liệu mà bạn muốn sắp xếp. Trong ví dụ 4 sẽ lấy kiểu String ở đầu mỗi record, ví dụ 5 sẽ lấy kiểu integer ở cuối mỗi record.

Ví dụ 4: StringSort.java

```
/*-------------------------------------------------
* StringSort.java
*
* Sort records that contain multiple Java
* data types. Sort using String type.
*
* Uses: Streams, Enumeration, RecordComparator
*
* No GUI interface, all output is to the console
*-------------------------------------------------*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
public class StringSort extends MIDlet
{
        private RecordStore rs = null; // Record store
        static final String REC_STORE = "db_3"; // Name of record store
        public StringSort()
```

```
{
        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());
                }
        }
}
/*-------------------------------------------------
* Create three arrays to write to record store
*-------------------------------------------------*/
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);
}
/*-------------------------------------------------
* Write to record store using streams.
*-------------------------------------------------*/
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());
        }
```

```
}
/*-------------------------------------------------
* Read from the record store using streams
*-------------------------------------------------*/
public void readStream()
{
        try
        {
                // Careful: Make sure this is big enough!
                // Better yet, test and reallocate if necessary
                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)
                {
                        ComparatorString comp = new ComparatorString();
                        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.compareStringClose();
                        // Free enumerator
                        re.destroy();
                }
                strmBytes.close();
                strmDataType.close();
        }
        catch (Exception e)
        {
                db(e.toString());
        }
}
/*-------------------------------------------------
* Simple message to console for debug/errors
* When used with Exceptions we should handle the
```

```
        * error in a more appropriate manner.
        *-----------------------------------------------*/
        private void db(String str)
        {
                System.err.println("Msg: " + str);
        }
}
/*-----------------------------------------------
| Compares two strings to determine sort order
| Each record passed in contains multiple Java data
| types - use only the String data for sorting
*-----------------------------------------------*/
class ComparatorString 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 compareStringClose()
        {
                try
                {
                        if (strmBytes != null)
                                strmBytes.close();
                        if (strmDataType != null)
                                strmDataType.close();
                }
                catch (Exception e)
                {
                }
        }
        public int compare(byte[] rec1, byte[] rec2)
        {
                String str1, str2;
                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
                        // Only need one read because the string to
                        // sort on is the first "field" in the record
                        strmBytes = new ByteArrayInputStream(rec1);
                        strmDataType = new DataInputStream(strmBytes);
                        str1 = strmDataType.readUTF();
                        // Read record #2
                        strmBytes = new ByteArrayInputStream(rec2);
                        strmDataType = new DataInputStream(strmBytes);
                        str2 = strmDataType.readUTF();
```

```
                    // Compare record #1 and #2
                    int result = str1.compareTo(str2);
                    if (result == 0)
                            return RecordComparator.EQUIVALENT;
                    else if (result < 0)
                            return RecordComparator.PRECEDES;
                    else
                            return RecordComparator.FOLLOWS;
            }
            catch (Exception e)
            {
                    return RecordComparator.EQUIVALENT;
            }
    }
}
```

Trường dữ liệu đầu tiên trong các record là kiểu string, - dùng làm tiêu chí sắp xếp. Trước hết ta lấy chuỗi cần so sánh trong dãy byte bằng hàm readUTF() , rồi dùng compareTo() trong class String để sắp xếp:

```
    // Read record #1
    // Only need one read because the string to
    // sort on is the first "field" in the record
    . . .
    str1 = strmDataType.readUTF();
    // Read record #2
    . . .
    str2 = strmDataType.readUTF();
    // Compare record #1 and #2
    int result = str1.compareTo(str2);
. . .
```

Output của ví dụ 4:

```
Record #1
Name: beauregard
Dog: true
Rank: 2
--------------------
Record #2
Name: duke
Dog: true
Rank: 3
--------------------
Record #3
Name: spike
Dog: true
Rank: 1
--------------------
Record #4
Name: tiger
Dog: false
Rank: 0
```

Ví dụ 5: integer sort

```
/*--------------------------------------------------
* IntSort.java
*
* Sort records that contain multiple Java
* data types. Sort using integer type.
*
* Uses: Streams, Enumeration, RecordComparator
*
* No GUI interface, all output is to the console
*--------------------------------------------------*/
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());
                }
        }
}
/*-------------------------------------------------
* Create three arrays to write to record store
*-------------------------------------------------*/
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);
}
/*-------------------------------------------------
* Write to record store using streams.
*-------------------------------------------------*/
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());
        }
}
/*-------------------------------------------------
* Read from the record store using streams
*-------------------------------------------------*/
public void readStream()
{
        try
        {
                // Careful: Make sure this is big enough!
                // Better yet, test and reallocate if necessary
                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());
                }
        }
        /*-------------------------------------------------
        * Simple message to console for debug/errors
        * When used with Exceptions we should handle the
        * error in a more appropriate manner.
        *-------------------------------------------------*/
        private void db(String str)
        {
                System.err.println("Msg: " + str);
        }
}
/*-------------------------------------------------
| Compares two integers to determine sort order
| Each record passed in contains multiple Java data
| types - use only the integer data for sorting
*-------------------------------------------------*/
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 ta vẫn sẽ dùng dữ liệu của ví dụ 4, 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:

```
Record #1
Name: tiger
Dog: false
Rank: 0
--------------------
Record #2
Name: spike
Dog: true
Rank: 1
--------------------
Record #3
Name: beauregard
Dog: true
Rank: 2
--------------------
Record #4
Name: duke
Dog: true
Rank: 3
--------------------
```

6. Searching with 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 sẽ được 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 6:

```
/*-------------------------------------------------
* SimpleSearch.java
*
* Display a Form and Textbox for searching records
* Each record contains a String object.
*
* Uses: Enumeration, RecordFilter
*-------------------------------------------------*/
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);
        // Define textfield, stringItem and commands

        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());
                }
        }
}
/*-------------------------------------------------
* Create array of data to write into record store
*-------------------------------------------------*/
public void writeTestData()
{
        String[] golfClubs = {
        "Wedge...good from the sand trap",
        "Truong dai hoc khoa hoc tu nhien",
        "Putter...only on the green",
        "Hoc mon LTUDM rat bo ich!"};
        writeRecords(golfClubs);
}
/*-------------------------------------------------
* Write to record store.
*-------------------------------------------------*/
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());
        }
}
/*-------------------------------------------------
* Search using enumerator and record filter
*-------------------------------------------------*/
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();
        }
}
/*-------------------------------------------------
* Simple message to console for debug/errors
* When used with Exceptions we should handle the
* error in a more appropriate manner.
*-------------------------------------------------*/
```

```
        private void db(String str)
        {
                System.err.println("Msg: " + str);
        }
}
/*-------------------------------------------------
* Search for text within a record
* Each record passed in contains only text (String)
*-------------------------------------------------*/
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:

Lưu ý ví dụ trên không phân biệt chữ hoa thường.

Tiếp theo đây chúng ta lại xét thêm một ví dụ về tìm kiếm record. Ví dụ 7:

```
/*-------------------------------------------------
* SearchStreams.java
*
* Display a Form and Textbox for searching records
* Each record contains multiple Java data types -
* (String, boolean and integer). Use the String
* data for searching
*
* Uses: Enumeration, RecordFilter
*-------------------------------------------------*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
public class SearchStreams 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
        private RecordStore rs = null; // Record store
        static final String REC_STORE = "db_7"; // Name of record store
```

```
public SearchStreams()
{
        display = Display.getDisplay(this);
        // Define textfield, stringItem and commands
        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());
                }
        }
}
/*-------------------------------------------------
* Create three arrays to write into record store
*-------------------------------------------------*/
public void writeTestData()
{
        String[] pets = {"duke - big, goofy golden retriever",
        "tiger - we found in a field",
        "spike - loves chasing car tires",
        "beauregard - barks at everything"};
        boolean[] dog = {true, false, true, true};
        int[] rank = {3, 0, 1, 2};
        writeStream(pets, dog, rank);
}
/*-------------------------------------------------
* Write to record store using streams.
*-------------------------------------------------*/
public void writeStream(String[] sData, boolean[] bData,int[] iData)
{
        try
        {
                // Only add the records once
                if (rs.getNumRecords() > 0)
                        return;
                // 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());
        }
}
/*-------------------------------------------------
* Search using enumerator and record filter
*-------------------------------------------------*/
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)
                        {
                        // Read from the specified byte array
                                ByteArrayInputStream strmBytes =
                                new ByteArrayInputStream(re.nextRecord());
                                // Read Java data types from the above byte array
                                DataInputStream strmDataType =
                                new DataInputStream(strmBytes);
                                // Show matching result in stringItem component on form
                                siMatch.setText(strmDataType.readUTF());
                                search.searchFilterClose(); // Close record filter
                                strmBytes.close(); // Close stream
```

```
                                        strmDataType.close(); // Close stream
                                        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();
        }
        }
        /*-------------------------------------------------
        * Simple message to console for debug/errors
        * When used with Exceptions we should handle the
        * error in a more appropriate manner.
        *-------------------------------------------------*/
        private void db(String str)
        {
                System.err.println("Msg: " + str);
        }
}
/*-------------------------------------------------
* Search for text within a record
* Each record passed in contains multiple Java data
* types (String, boolean and integer)
*-------------------------------------------------*/
class SearchFilter implements RecordFilter
{
        private String searchText = null;
        // Read from a specified byte array
        private ByteArrayInputStream strmBytes = null;
        // Read Java data types from the above byte array
        private DataInputStream strmDataType = null;
        public SearchFilter(String searchText)
        {
        // This is the text to search for
                this.searchText = searchText.toLowerCase();
        }
        // Cleanup
        public void searchFilterClose()
```

```
        {
                try
                {
                        if (strmBytes != null)
                        strmBytes.close();
                        if (strmDataType != null)
                        strmDataType.close();
                }
                catch (Exception e)
                {
                }
        }
        public boolean matches(byte[] candidate)
        {
                String str = null;
                try
                {
                        strmBytes = new ByteArrayInputStream(candidate);
                        strmDataType = new DataInputStream(strmBytes);
                        // Although 3 pieces of data were written to
                        // the record (String, boolean and integer)
                        // we only need one read because the string to
                        // search is the first "field" in the record
                        str = strmDataType.readUTF().toLowerCase();
                }
                catch (Exception e)
                {
                        return false;
                }
                // Look for a match
                if (str != null && str.indexOf(searchText) != -1)
                        return true;
                else
                        return false;
        }
}
```

Cùng có chức năng tìm kiếm nhưng ví dụ 7 khác ví dụ 6 ở chỗ dữ liệu lưu vào record. Ở ví dụ này dữ liệu lưu vào record dưới dạng dãy byte, và trong dãy byte này ta lại lưu nhiều trường dữ liệu:

```
strmDataType.writeUTF(sData[i]); // Write Strings
strmDataType.writeBoolean(bData[i]); // Write booleans
strmDataType.writeInt(iData[i]); // Write integers
```

Trong hàm searchRecordStore() ta tạo một bộ tìm kiếm và enumerator. Phương thức matches() (của class SearchFilter) sẽ được gọi bởi enumerator và được áp dụng cho mỗi record trong RecordStore. Để đọc được đúng dữ liệu cần dùng ta cần dùng hai stream – một dùng để đọc dãy byte trong record và một dùng để đọc đúng kiểu dữ liệu trong dãy byte đó.

```
strmBytes = new ByteArrayInputStream(candidate);
strmDataType = new DataInputStream(strmBytes);
str = strmDataType.readUTF().toLowerCase();
```

Sau đó chuỗi này sẽ được so sánh với searchText:
```
if (str != null && str.indexOf(searchText) != -1)
return true;
else
return false;
```

Output:

7. Notification of Changes with 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 8: sử dụng RecordListener

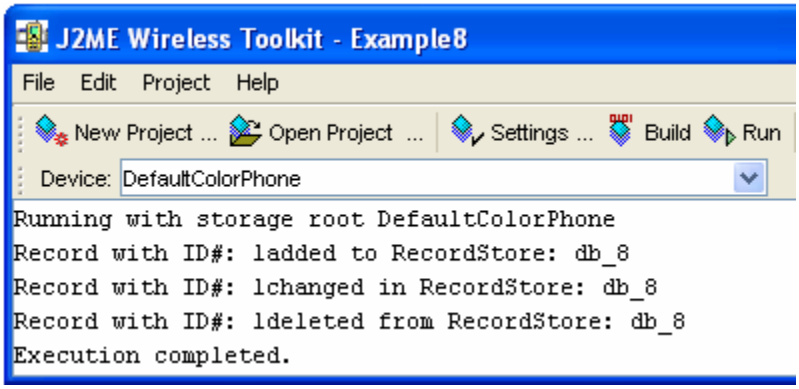
```
/*-------------------------------------------------
* RmsListener.java
*
* Test the RMS listener methods
*
* No GUI interface, all output is to the console
*-------------------------------------------------*/
import java.io.*;
import javax.microedition.midlet.*;
import javax.microedition.rms.*;
public 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());
                }
        }
        /*-------------------------------------------------
        * Simple message to console for debug/errors
        * When used with Exceptions we should handle the
        * error in a more appropriate manner.
        *-------------------------------------------------*/
        public void db(String str)
        {
                System.err.println("Msg: " + str);
        }
}
        /*-------------------------------------------------
        * Listen for updates to the record store
        *-------------------------------------------------*/
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);
                }
        }
}
```

Output:



8. Exception Handling

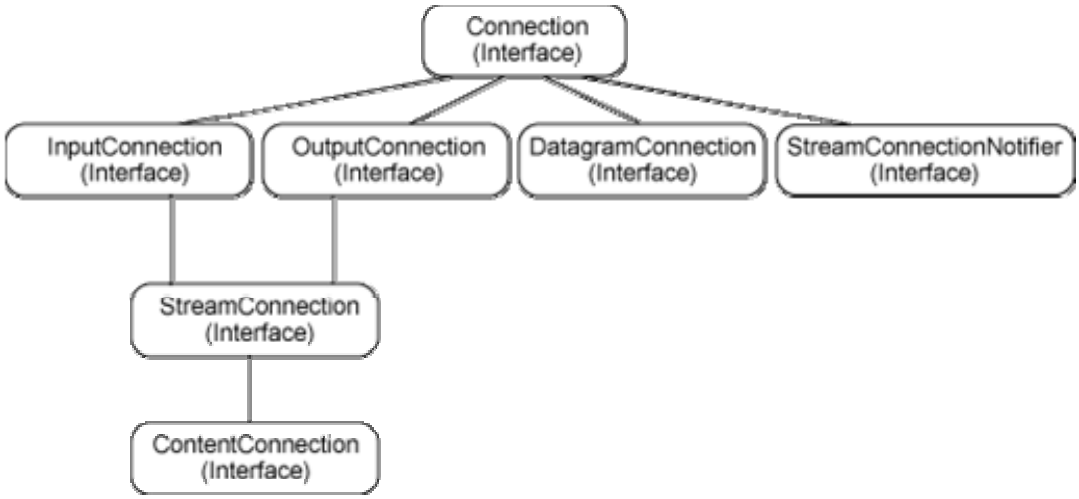
Các phương thức trong API của RMS ngoài việc phát sinh các ngoại lệ thông thường đến môi trường chạy (runtime enviroment). RMS còn định nghĩa thêm các ngoại lệ trong gói *javax.microedition.rms* như sau:

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

VI. The Generic Connection Framework

Với kích thước hơn 200 kb và hơn 100 class và interfaces trong gói java.io, java.net của J2SE sẽ chiếm hầu hết bộ nhớ vốn dĩ đã nhỏ bé của những thiết bị di động. Do đó Sun không thể kế thừa những gói này vào trong J2ME, mà họ đã xây dựng một chuẩn là Generic Connection Framework (GCF). GCF sẽ giúp cho các thiết bị di động có thể truy xuất được các tài nguyên mạng, cũng như các tài nguyên khác mà địa chỉ của chúng được xác định bằng URL.

Generic Connection Framework bao gồm một tập hợp các interfaces được khai báo trong package javax.microedition.io. Hình vẽ sau thể hiện mối quan hệ giữa các interfaces:



Có một class chính là Connector và 7 interfaces đã được định nghĩa trong GCF. Từ hình vẽ ta dễ dàng nhận thấy GCF hỗ trợ kết nối dạng datagram(packet) và stream. Class Connector được dùng để mở kết nối đến một tài nguyên nào đó thông qua phương thức open được khai báo như sau:

```
Connector.Open("protocol:address;parameters");
```

1. Những protocol được hỗ trợ trong GCF

GCF hỗ trỡ nhiều loại protocol khác nhau. Khi có yêu cầu tạo một kết nối, class Connector sẽ sử dụng phương thức Class.forName() để tìm kiếm một class phù hợp với protocol đó. Nếu tìm thấy một đối tượng sẽ được trả về và thực thi interface Connection (hình vẽ). Dưới đây là cách để mở các kết nối khác nhau:

```
Connector.Open("socket://www.corej2me.com.com:55");
Connector.Open("http://www.corej2me.com");
Connector.Open("datagram://www.corej2me.com:1000");
Connector.Open("file://makefile.txt");
```

Mở một kết nối

Để tạo kết nối, GCF đã cung cấp cho ta đến 7 phương thức:

```
Connector  (public class Connector)
  public static Connection open(String name)
```

```
public static Connection open(String name)
public static Connection open(String name, int mode, boolean timeouts)
public static DataInputStream openDataInputStream(String name)
public static DataOutputStream openDataOutputStream(String name)
public static InputStream openInputStream(String name)
public static OutputStream openOutputStream(String name)
```

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);
}
```

Tuy nhiên, ngoài class ContentConnection, ta cũng có thể mở một kết nối trực tiếp bằng InputStream. Sau đây là đoạn code cho phép tải về một bức ảnh và sau đó tạo lại bức ảnh đó:

```
InputStream iStrm = (InputStream) Connector.openInputStream(url);
Image img = 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
  img = Image.createImage(imageData, 0, imageData.length);
}
finally
{
  // Clean up
  if (iStrm != null)
    iStrm.close();
}
```

Lưu ý, trong đoạn code trên do không sử dụng class ContentConnection, nên ta sẽ không biết được kích thước của dữ liệu sẽ tải về. Để khắc phục vấn đề này, người ta sẽ dùng ByteArrayOutputStream để lưu dữ liệu tải về.

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:

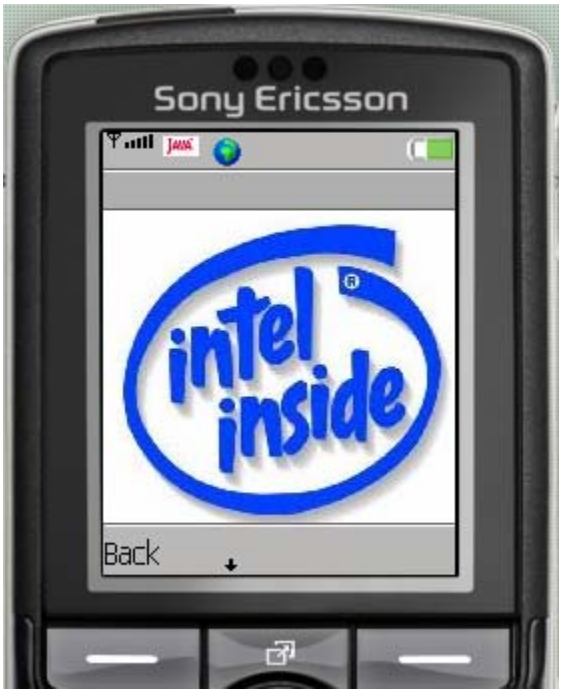
```
/*-------------------------------------------------
* DownloadImage.java
*
* Download and view a png file
*-------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;

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)
{ }

/*-------------------------------------------------
* Process events
*-------------------------------------------------*/
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);
    }
  }

  /*-------------------------------------------------
  * Open connection and download png into a byte array.
  *-------------------------------------------------*/
  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

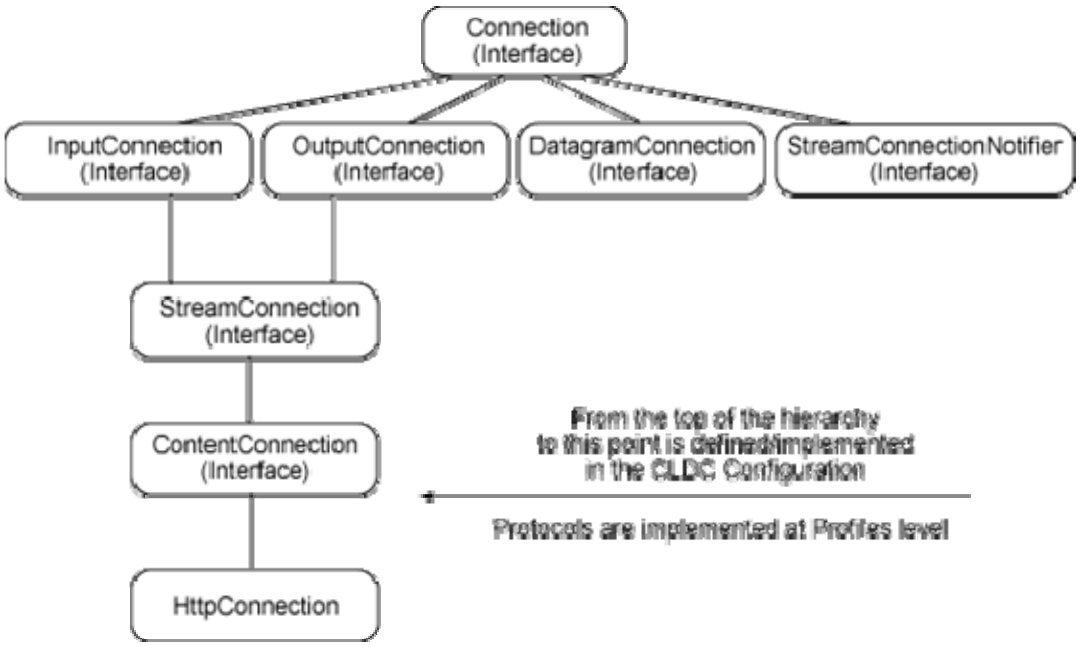
Sau khi tải về, hình ảnh sẽ được hiển thị



2. Hỗ trợ giao thức HTTP trong MIDP

Đến đây thì bạn dã biết được rằng GCF hỗ trợ nhiều kiểu kết nối và đã phát triển ứng dụng MIDlet tải về và hiển thị hình ảnh. Trong phần này, sẽ bàn sâu hơn về sự hỗ trợ HTTP

của GCF. Đầu tiên, hãy cập nhật lại sơ đồ quan hệ giữa các interfaces trong GCF để thấy được vị trí của http:



Đặc tả của phiên bản MIDP 1.0 chỉ hỗ trợ HTTP, trong khi đó phiên bản hiện tại MIDP 2.0 hỗ trợ HTTP và HTTPS, cung cấp khả năng bảo mật tốt hơn. Các hàm API được khai báo trong HttpConnection (cho HTTP) và trong HttpConnections (cho HTTP và HTTPS).

a) Request and response protocols

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 sẽ được 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 thể đặt 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 từ client 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 để đọc kết quả trả về từ server.

b) 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)

Ví dụ: Chúng ta sẽ tạo một MIDlet sử dụng HttpConnection để tải về và hiển thị nội dung của file text. Qua ví dụ bạn sẽ hiểu cách client gửi request và lấy response của server. MIDlet cũng sử dụng một số phương thức trong HttpConnection để thu thập thông tin về server: port, content type .. .

```
/*-------------------------------------------------
* FileViewer.java
*-------------------------------------------------*/
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class FileViewer extends MIDlet implements CommandListener
{
  private Display display;
  private Form fmMain;
  private Command cmExit;
  private Command cmView;
  private String url = "http://localhost/text.txt";

  public FileViewer()
  {
    display = Display.getDisplay(this);

    // Create exitcommand
    cmExit = new Command("Exit", Command.EXIT, 1);
    cmView = new Command("View", Command.SCREEN, 2);

    // Create the form and add commands
    fmMain = new Form("File Viewer");
    fmMain.addCommand(cmExit);
    fmMain.addCommand(cmView);

    fmMain.setCommandListener(this);
  }

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

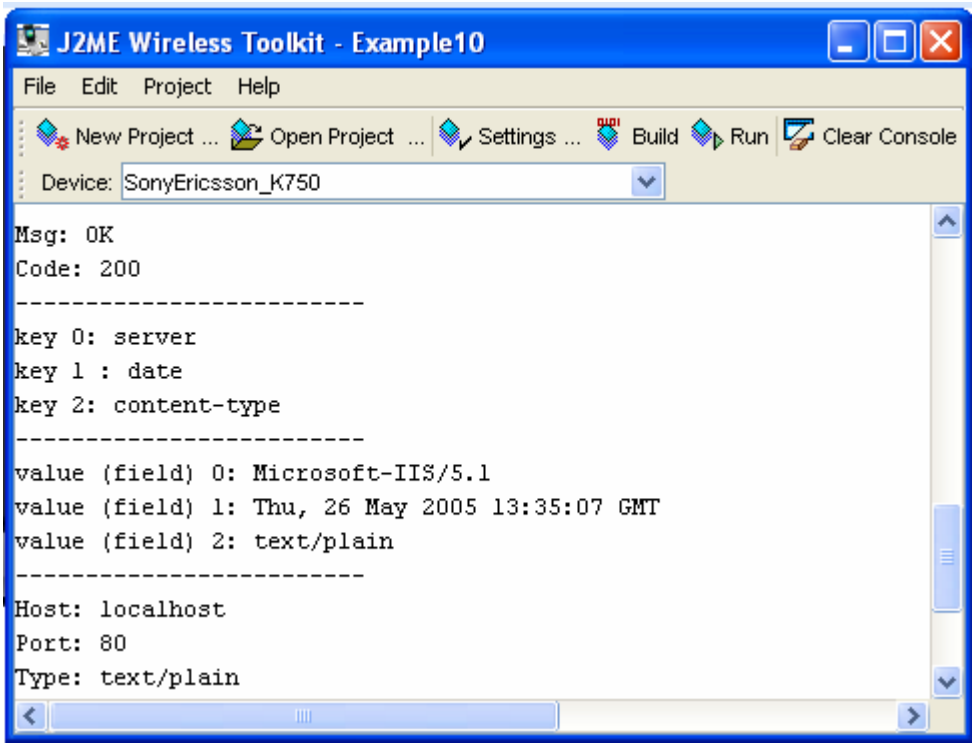
```
/*------------------------------------------------
* Process events
*------------------------------------------------*/
public void commandAction(Command c, Displayable s)
{
  // If the Command button pressed was "Exit"
  if (c == cmExit)
  {
    destroyApp(false);
    notifyDestroyed();
  }
  else if (c == cmView)
  {
    // Download image and place on the form
    try
    {
      String str;
      if ((str = readFile()) != null)
      {
        // Delete form contents
        for (int i = fmMain.size(); i > 0; i--)
          fmMain.delete(0);

        // Append downloaded string
        fmMain.append("" + str);
      }
    }
    catch (Exception e)
    {
      System.err.println("Msg: " + e.toString());
    }
  }
}

/*------------------------------------------------
* Read file
*------------------------------------------------*/
private String readFile() throws IOException
{
  HttpConnection http = null;
  InputStream iStrm = null;
  String str = null;

  try
  {
    // Create the connection
    http = (HttpConnection) Connector.open(url);

    //----------------
    // Client Request
```

```
//----------------
// 1) Send request method
http.setRequestMethod(HttpConnection.GET);

// 2) Send header information (this header is optional)
http.setRequestProperty("User-Agent", "Profile/MIDP-1.0 Configuration/CLDC-1.0");

// 3) Send body/data - No data for this request


//----------------
// Server Response
//----------------
System.out.println("url: " + url);
System.out.println("------------------------");

// 1) Get status Line
System.out.println("Msg: " + http.getResponseMessage());
System.out.println("Code: " + http.getResponseCode());
System.out.println("------------------------");

// 2) Get header information
if (http.getResponseCode() == HttpConnection.HTTP_OK)
{
  System.out.println("key 0: " + http.getHeaderFieldKey(0));
  System.out.println("key 1 : " + http.getHeaderFieldKey(1));
  System.out.println("key 2: " + http.getHeaderFieldKey(2));
  System.out.println("------------------------");

  System.out.println("value (field) 0: " + http.getHeaderField(0));
  System.out.println("value (field) 1: " + http.getHeaderField(1));
  System.out.println("value (field) 2: " + http.getHeaderField(2));
  System.out.println("------------------------");

  // 3) Get data (show the file contents)
  iStrm = http.openInputStream();
  int length = (int) http.getLength();
  if (length != -1)
  {
    // Read data in one chunk
    byte serverData[] = new byte[length];
    iStrm.read(serverData);
    str = new String(serverData);
  }
  else  // Length not available...
  {
    ByteArrayOutputStream bStrm = new ByteArrayOutputStream();

    // Read data one character at a time
    int ch;
    while ((ch = iStrm.read()) != -1)
```

```
          bStrm.write(ch);

        str = new String(bStrm.toByteArray());
        bStrm.close();
      }

      //----------------------------
      // Show connection information
      //----------------------------
      System.out.println("Host: " + http.getHost());
      System.out.println("Port: " + http.getPort());
      System.out.println("Type: " + http.getType());
    }
  }
  finally
  {
    // Clean up
    if (iStrm != null)
      iStrm.close();
    if (http != null)
      http.close();
  }

  return str;

 }

 public void pauseApp()
 { }

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

Output

Console output



3. Accessing a Java servlet

Ví dụ dưới đây sẽ chỉ cho bạn từng bước cách truy truy xuất java servlet. Bước đầu, client sẽ truyền tham số cho servlet yêu cầu trả về ngày tháng theo định dạng. Servlet sẽ trà về kết quả, và sẽ hiện lên màn hình.

Đây là kết quả cho chuỗi: “yyyy.MM.dd+'at'+hh:mm:ss+zzz"

Trong hình trên, ta thấy ở phần kết quả có time zones là ICT, do ta đã truyền tham số “zzz”, nếu chuyển thành “zzzz” để lấy full text của time zones thì kết quả sẽ thay đổi như sau:

Calling Servlet

Yêu cầu gọi servlet sẽ được khai báo bên trong commandAction(), và sẽ được thực hiện khi button cmRqst được kích hoạt:

```
/*------------------------------------------------
* Call the servlet
*------------------------------------------------*/
public void commandAction(Command c, Displayable s)
{
  if (c == cmRqst)
  {
    try
    {
      serverMsg = null;
      callServlet();
      if (serverMsg != null)
        fmMain.append(serverMsg);
    }
    catch (Exception e)
    {
      System.err.println("Msg: " + e.toString());
    }
  }
  else if (c == cmExit)
  {
    destroyApp(false);
    notifyDestroyed();
  }
}
```

Dưới đây là đoạn code cho phương thức callServlet(), chú ý rằng request của client sử dụng request method là GET,:

```
/*------------------------------------------------
* Call the servlet
*------------------------------------------------*/
private void callServlet() throws IOException
{
  HttpConnection http = null;
  InputStream iStrm = null;
  boolean ret = false;

  // Examples - Data is passed at the end of url for GET
String url = "http://localhost:8080/j2me/dateformatservlet?format=MMMM.dd.yyyy+'-
'+hh:mm+aa";
//                                                    String url =
"http://localhost:8080/j2me/dateformatservlet?format=yyyy.MM.dd+'at'+hh:mm:ss+zzz";
  try
  {
    http = (HttpConnection) Connector.open(url);

    //----------------
```

```
            // Client Request
            //----------------
            // 1) Send request method
            http.setRequestMethod(HttpConnection.GET);
            // 2) Send header information - none
            // 3) Send body/data -  data is at the end of URL

            //----------------
            // Server Response
            //----------------
            iStrm = http.openInputStream();
            // Three steps are processed in this method call
            ret = processServerResponse(http, iStrm);
          }
          finally
          {
            // Clean up
            if (iStrm != null)
              iStrm.close();
            if (http != null)
              http.close();
          }

          // Process request failed, show alert
          if (ret == false)
            showAlert(errorMsg);
        }
```

Code cho servlet, hầu hết các phương thức đều được biết trong doPost(), biến format dùng để lưu request từ client, sau đó servlet sẽ gọi SimpleDateFormat để tạo ra kết quả phù hợp với request:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.Date;
import java.text.SimpleDateFormat;
import java.util.TimeZone;
import java.util.Locale;

public class DateFormatServlet extends HttpServlet
{
    public void init(ServletConfig config) throws ServletException
    {
      super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
```

```
    {
      doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException
    {
      String format = request.getParameter("format");
      SimpleDateFormat simpleDate = new SimpleDateFormat(format);
      Date dt = new Date();

      PrintWriter out = response.getWriter();
      response.setContentType("text/html");
      out.println(simpleDate.format(dt));
      out.close();
    }

    public String getServletInfo()
    {
      return "DateFormatServlet";
    }
}
```

Để MIDlet nhận kết quả trả về từ servlet, trước hết phải kiểm tra status line với server và có kết trả về kết quả hay không (HTTP_OK). Do không có thông tin header nên sẽ tiến hành lấy kết quả từ input stream:

```
/*-------------------------------------------------
* Process a response from a server
*-------------------------------------------------*/
private boolean processServerResponse(HttpConnection http, InputStream iStrm)
throws IOException
{
  //Reset error message
  errorMsg = null;

  // 1) Get status Line
  if (http.getResponseCode() == HttpConnection.HTTP_OK)
  {
    // 2) Get header information - none

    // 3) Get body (data)
    int length = (int) http.getLength();
    String str;
    if (length != -1)
    {
      byte servletData[] = new byte[length];
      iStrm.read(servletData);
      str = new String(servletData);
    }
```

```
    else  // Length not available...
    {
      ByteArrayOutputStream bStrm = new ByteArrayOutputStream();

      int ch;
      while ((ch = iStrm.read()) != -1)
        bStrm.write(ch);

      str = new String(bStrm.toByteArray());
      bStrm.close();
    }

   // Save the server message
   serverMsg = str;
   return true;

  }
  else
    // Use message from the servlet
    errorMsg = new String( http.getResponseMessage());

  return false;
}
```

Phụ lục

Bảng 1: Tổng số class và interface trong CLDC		
Tên gói	**Miêu tả**	**Số lượng class và interface**
java.io	Quản lý việc xuất nhập của hệ thống	18
java.lang	Các lớp Java cơ bản	38
java.util	Các lớp hỗ trợ việc tính thời gian, quản lý tập hợp v.v	10
javax.microedition	Quản lý kết nối	10

Bảng 2: Các lớp bổ sung trong MIDP 1.0		
Tên gói	**Miêu tả**	**Số lượng class và interface**
javax.microedition.rms	Quản lý việc lưu trữ dữ liệu	10
javax.microedition.midlet	Giao diện trung gian giữa Midlet và môi trường Midlet đang chạy	2
javax.microedition.io	Hỗ trợ HTTP	1
javax.microedition.lcdui	Hỗ trợ giao diện người dùng	24

Bảng 3: Tổng số class và interface trong CDC		
Tên gói	**Miêu tả**	**Số lượng class và interface**
java.io	Quản lý việc xuất nhập của hệ thống	62
java.lang	Các lớp Java cơ bản	77
java.lang.ref	Các lớp hỗ trợ bộ thu gom "rác" của JVM	5
java.lang.reflect	Hỗ trợ tính năng reflection trong Java	12
java.math	Hỗ trợ các phép tính trong toán học	1
java.net	Hỗ trợ mạng	23
java.security	Cung cấp các tính năng bảo mật	36
java.security.cert	Hỗ trợ các chứng thực trong bảo mật	4
java.text	Cung cấp các lớp hỗ trợ thao tác trên text	13
java.util	Các lớp hỗ trợ việc tính thời gian, quản lý tập hợp v.v	47
java.util.jar	Hỗ trợ thao tác trên file Jar	7
java.util.zip	Hỗ trợ thao tác trên file Zip	9
javax.microedition	Quản lý kết nối	10

Tham khảo:
http://java.sun.com/j2me/
http://www.cs.armstrong.edu/burge/pdf/huc/01_J2ME_Overview.pdf
http://www.onjava.com/pub/a/onjava/2001/03/08/J2ME.html
http://developers.sun.com/techtopics/mobility/midp/articles/wtoolkit/
CLDC Specification (JSR-30)
http://www.jcp.org/jsr/detail/30.jsp
CLDC Next Generation (JSR-139)
http://www.jcp.org/jsr/detail/139.jsp
MIDP Specification (JSR-37)
http://www.jcp.org/jsr/detail/37.jsp
Mobile Information Device Next Generation (JSR-118)
http://www.jcp.org/jsr/detail/132.jsp
PDA Profile Specification (JSR-75)
http://www.jcp.org/jsr/detail/75.jsp
CDC Specification (JSR-36)
http://www.jcp.org/jsr/detail/36.jsp
Foundation Profile Specification (JSR-46)
http://www.jcp.org/jsr/detail/46.jsp
Personal Profile Specification (JSR-62)
http://www.jcp.org/jsr/detail/62.jsp
RMI Profile Specification (JSR-66)
http://www.jcp.org/jsr/detail/66.jsp
Java Game Profile (JSR-134)
http://www.jcp.org/jsr/detail/134.jsp
J2ME Multimedia API Specification (JSR-135)
http://www.jcp.org/jsr/detail/135.jsp
J2ME White Papers
KVM
http://java.sun.com/products/cldc/wp
Applications for Mobile Devices
http://java.sun.com/j2me/docs/pdf/midpwp.pdf
J2ME Technologies
J2ME
http://java.sun.com/j2me
CLDC and KVM
http://java.sun.com/products/cldc
MIDP
http://java.sun.com/products/midp
MIDP for Palm OS
http://java.sun.com/
CDC and CVM
http://java.sun.com/products/cdc

J2ME Development Kits
J2ME Wireless Toolkit
http://java.sun.com/products/j2mewtoolkit
Metrowerks CodeWarrior for J2ME
http://www.metrowerks.com/desktop/java
Zucotto WHITEboard SDK
http://www.zucotto.com/whiteboard/index.html
RIM BlackBerry Java IDE
http://developers.rim.net/tools/jde/index.shtml
J2ME Developer Resources
Java Developer Connection
http://developer.java.sun.com/developer
Java Wireless Developer Initiative
http://java.sun.com/wireless
Wireless Developer Network
http://www.wirelessdevnet.com
Micro Java Network
http://www.microjava.com
Java Mobile Community
http://www.javamobile.org
Java Enabled Phones and PDAs
http://www.javamobiles.com
KVM World
http://www.kvmworld.com
KVM-Interest Mailing List Archive
http://archives.java.sun.com/kvm-interest.html
XML Parsers for J2ME
TinyXML Parser
http://gibaradunn.srac.org/tiny
NanoXML Parser
http://nanoxml.sourceforge.net