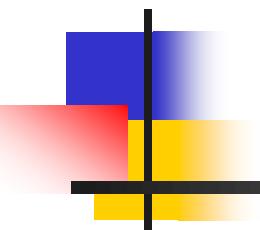
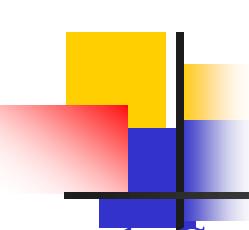


LẬP TRÌNH J2ME CHO THIẾT BỊ DI ĐỘNG

PHẦN 1





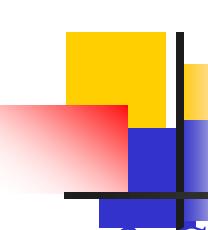
Hướng tự nghiên cứu (1)

1. Cơ sở về hệ thống thông tin di động (liên quan đến IT)

- Mạng GSM (Global System for Mobile Communication - Hệ thống thông tin di động toàn cầu)
- Kiến trúc mạng GSM và các thế hệ của nó
- Khía cạnh liên kết sóng vô tuyến (radio)
- Khía cạnh mạng

2. Hai công nghệ IEEE802.11 và Bluetooth

- Khái niệm IEEE802.11 (ở mạng cục bộ) và Bluetooth
- Các mô hình OSI
- Bluetooth và 802.11 dùng các scenario
- Bluetooth Multiplayer Games Framework (BluetoothMGF)
- Các vấn đề khác



Hướng tự nghiên cứu (2)

3. Giao thức ứng dụng không dây (Wireless Application Protocol – WAP)

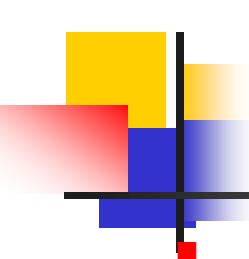
- Giới thiệu giao thức WAP
- Cổng WAP
- WML (Wireless Markup Language)/ WML script
- Web Service

4. VoiceXML

5. Các sản phẩm

- Tự mình thiết kế và xây dựng
- Nghiên cứu các sản phẩm sẵn có sau đó phát triển lên

6. Các lựa chọn khác



1.Giới thiệu về J2ME

Lịch sử

J2ME được phát triển từ kiến trúc Java Card, Embedded 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.

■ Lý do chọn J2ME

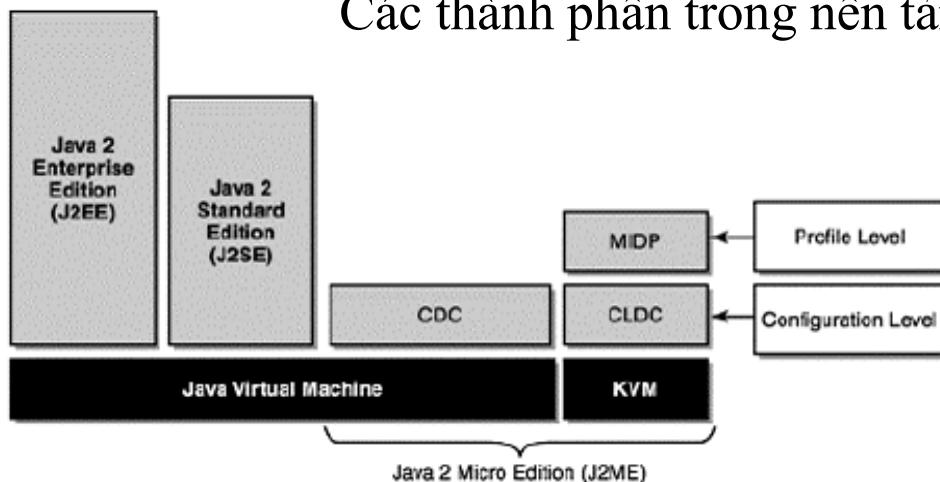
Java ban đầu được thiết kế dành cho các máy với tài nguyên bộ nhớ hạn chế.

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

- Các loại thẻ cá nhân như Java Card
- Máy điện thoại di động
- Máy PDA (Personal Digital Assistant - thiết bị trợ giúp cá nhân)
- Các hộp điều khiển dành cho tivi, thiết bị giải trí gia dụng ...

Kiến trúc của J2ME (1)

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. Hiện nay Sun đã đưa ra 2 dạng Configuration:

Kiến trúc của J2ME (2)

- 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 nhằm 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 ...
- Định nghĩa về Profile: Profile mở rộng Configuration bằng cách thêm vào các class để hỗ 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)

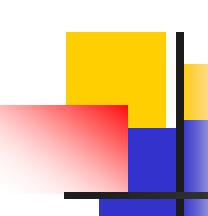
Giới thiệu MIDP (1)

■ 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.

■ Những chức năng MIDP không thực hiện được:

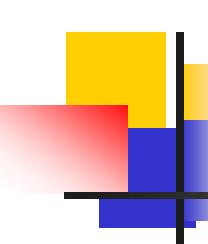
- Phép tính dấu phẩy động (floating point): Phép tính này đòi hỏi rất nhiều tài nguyên CPU và phần lớn các CPU cho các thiết bị di động không hỗ trợ phép tính này, do đó MIDP cũng không có.
- Bộ nạp lớp (Class Loader).
- Hỗ trợ từ khóa finalize() như trong J2SE: Việc “dọn dẹp” tài nguyên trước khi nó 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.



Giới thiệu MIDP (2)

Những chức năng MIDP cung cấp :

- Các lớp và kiểu dữ liệu: các lớp trong gói java.util như Stack, Vector, Hastable cũng như Enumeration.
- Hỗ trợ đối tượng Display: một chương trình MIDP sẽ hỗ trợ duy nhất một đối tượng Display, 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
- 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



Giới thiệu MIDP (3)

- Thêm các API hỗ trợ Multimedia. Cải tiến hấp dẫn nhất của MIDP 2.0 là tập các API media. 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. 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.

Môi trường phát triển J2ME (1)

- Một môi trường phát triển tích hợp (IDE) nhằm để cải thiện năng suất của lập trình viên bằng cách cung cấp một tập các công cụ lập trình tích hợp thông qua một giao diện người dùng đồ họa (GUI)
- Một IDE cho J2ME cần phải cung cấp các tiện ích sau:
 - Quản lý project - Quản lý các tập tin nguồn và các thông số MIDlet
 - Trình soạn thảo - Soạn thảo mã nguồn và các tài nguyên
 - Build (Biên dịch)
 - obfuscate (tùy chọn): xóa loại bỏ các thông tin không cần thiết trong class (như tên của các biến cục bộ, các lớp, phương thức,...). Ngoài việc bảo vệ mã nguồn, obfuscate còn giảm kích thước của các tập tin class, làm cho kích thước của tập tin JAR cũng giảm đi...
 - pre-verify (tiền kiểm tra)
 - Đóng gói (package) - Đóng gói các MIDlet thành các tập tin JAR và JAD
 - Giả lập (emulation) - Thực thi các MIDlet với một trình giả lập
 - Gỡ rối (debugger) - Gỡ rối MIDlet

Môi trường phát triển J2ME (2)

■ Các J2ME IDE phổ biến và nổi tiếng sau:

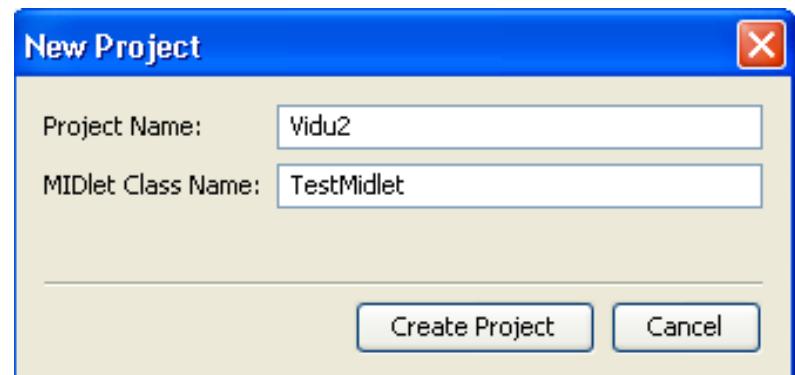
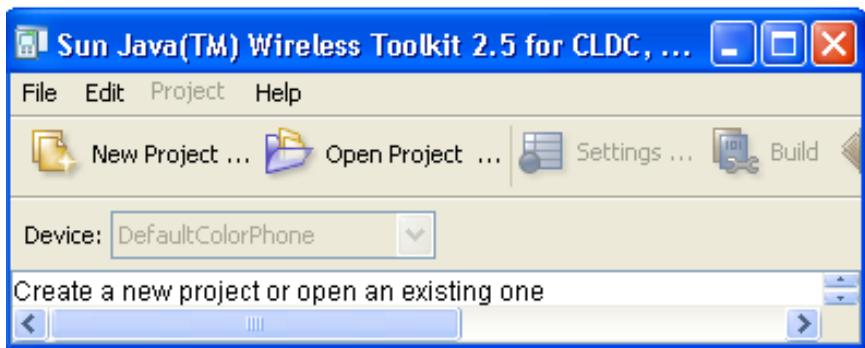
- Sun J2ME Wireless Toolkit 2.5
- Borland Jbuilder
- NetBeans IDE
- IntelliJ IDEA 3089
- Eclipse với EclipseME plug-in

■ Sun J2ME Wireless Toolkit 2.5 (WTK)

- WTK là một bộ công cụ phát triển Java J2ME (Java Development Kit - JDK) cung cấp cho các lập trình viên môi trường giả lập, công cụ, tài liệu và các ví dụ cần thiết để phát triển các ứng dụng MIDP.
- WTK không phải là một IDE hoàn chỉnh, vì nó đã bỏ các tính năng soạn thảo và gỡ rối vốn được xem là bắt buộc phải có trong một IDE. Nhưng KToolbar, được cung cấp trong bộ WTK là một môi trường phát triển tối thiểu cung cấp một GUI dành cho việc biên dịch, đóng gói và thực thi các ứng dụng MIDP.
- WTK 2.5 cũng cung cấp các bộ giả lập đã được cải tiến với các tính năng giả lập, monitor và debug mới. Có một cơ chế được thêm vào tiến trình build của KToolbar để cho phép việc tích hợp và thực thi bộ obfuscate Java byte code khi đóng gói MIDlet suite.

Môi trường phát triển J2ME (3)

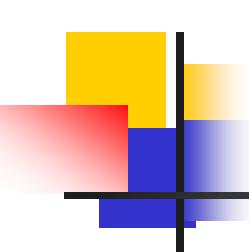
- Cài đặt bộ J2SE vào máy tính, địa chỉ tải J2SE <http://java.sun.com/j2se/>
- Cài đặt J2ME Wireless Toolkit, Địa chỉ <http://java.sun.com/j2me/download.html>



- Chọn "New Project..." để tạo một project mới.
- Nhập tên project (tên của file JAR và tên của thư mục project mới), nhập tên của MIDlet class (là main class của ứng dụng)
- Thư mục : “C:\WTK25-Beta2\apps\Vidu2\src”, đây sẽ là nơi chứa source của ứng dụng. Có thể dùng bất kỳ chương trình soạn thảo văn bản nào để soạn code.
- Tiến hành build và run chương trình

Môi trường phát triển J2ME (4)

- Nhấn vào "Settings..." trên toolbar để vào menu cấu hình cho project.
- Đừng để ý đến trường "MIDlet-Jar-Size" (với giá trị là "100" bytes), Chúng ta sẽ làm cho giá trị tự được thiết lập đúng.
- Chọn MIDlets tab trong cửa sổ dialog cấu hình của porject.
- Chọn hàng duy nhất trong bảng ("MIDlet-1") để làm nổi nó và chọn. Sau đó nhấn vào nút "Edit".
- Xoá trường "Icon" nếu không có tập tin *.PNG để đặt vào tập tin JAR.
- Chấp nhận các thay đổi.
 - (1) Project --> Clean: Xoá tất cả tập tin *.class.
 - (2) Build : Build tất cả tập tin *.class và preverify.
 - (3)Project --> Package --> Create Package: Sinh ra tập tin *.JAR và *.JAD.
Khi làm 3 bước trên, tập tin *.JAR và *.JAD kết quả đã có thể sẵn sàng được thực thi trong chương trình mô phỏng, hay đưa lên WWW site để download. Trường kích thước của *.JAR trong tập tin *.JAD sẽ tự đúng.
- Đừng quên thực hiện bước 3 mỗi khi rebuild

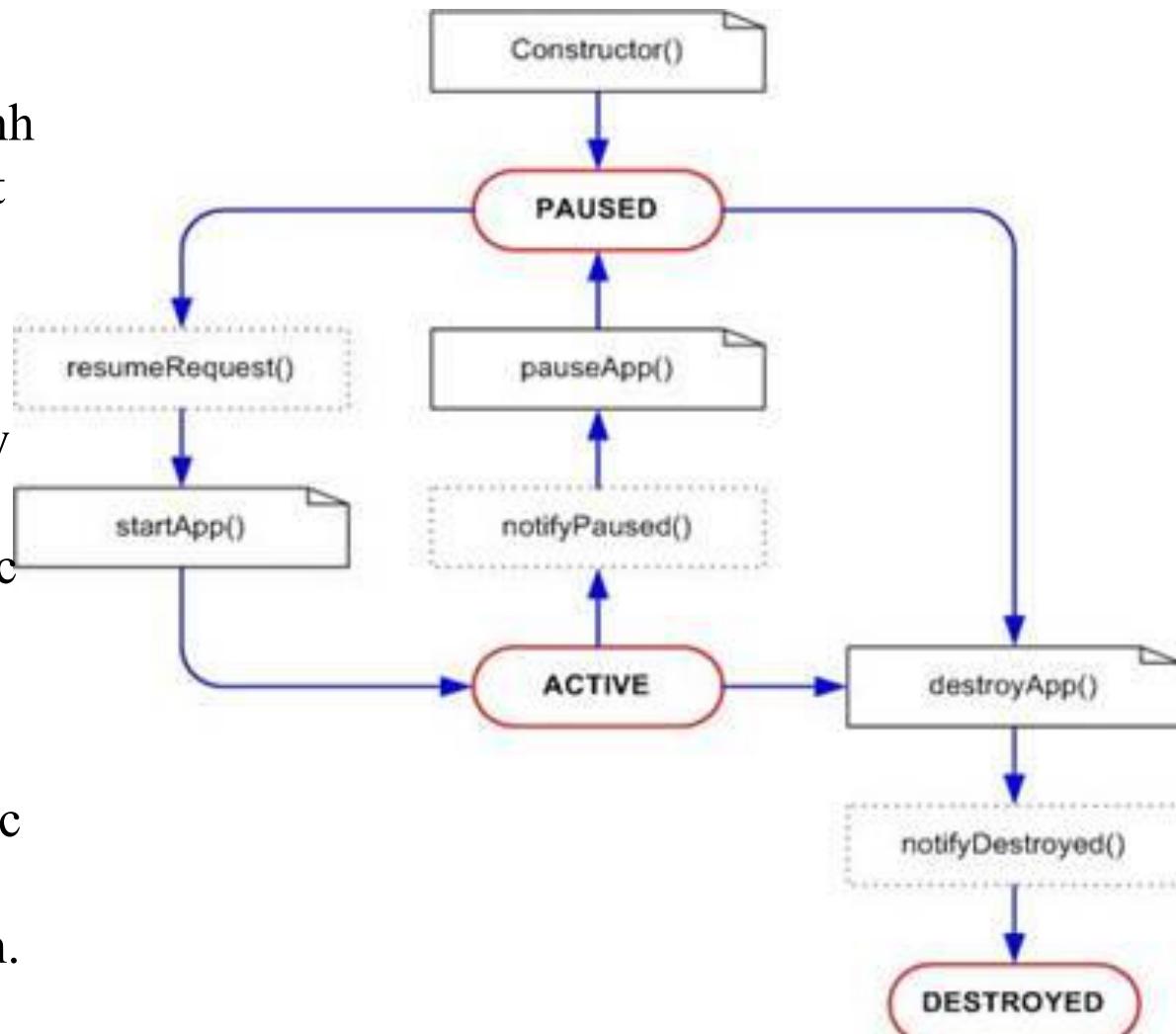


Chương trình đơn giản : Hello(Lời chào)

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
public class TestMidlet extends MIDlet {
    private Form mForm;
    public TestMidlet() {
        mForm = new Form("Lap trinh voi J2ME");
        mForm.append(new StringItem(null, "Hello world!, MIDP!"));
    }
    public void startApp() {
        Display.getDisplay(this).setCurrent(mForm);
    }
    public void pauseApp() {}
    public void destroyApp(boolean unconditional) {}
}
```

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,
 - 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.



import javax.microedition.lcdui & midlet

■ import javax.microedition.lcdui

- Interfaces: Choice, CommandListener, ItemCommandListener, ItemStateListener
- Classes: Alert, AlertType, Canvas, ChoiceGroup, Command, CustomItem, DateField, Display, Displayable, Font, Form, Gauge, Graphics, Image, ImageItem, Item, List, Screen, StringItem, TextBox, TextField, Ticker..
- Ví dụ ta có thể khai báo:

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

Hoặc khai chi tiết

```
import javax.microedition.lcdui.Display;  
import javax.microedition.lcdui.Form;  
import javax.microedition.lcdui.StringItem;  
import javax.microedition.lcdui.TextField; //.....
```

■ import javax.microedition.midlet

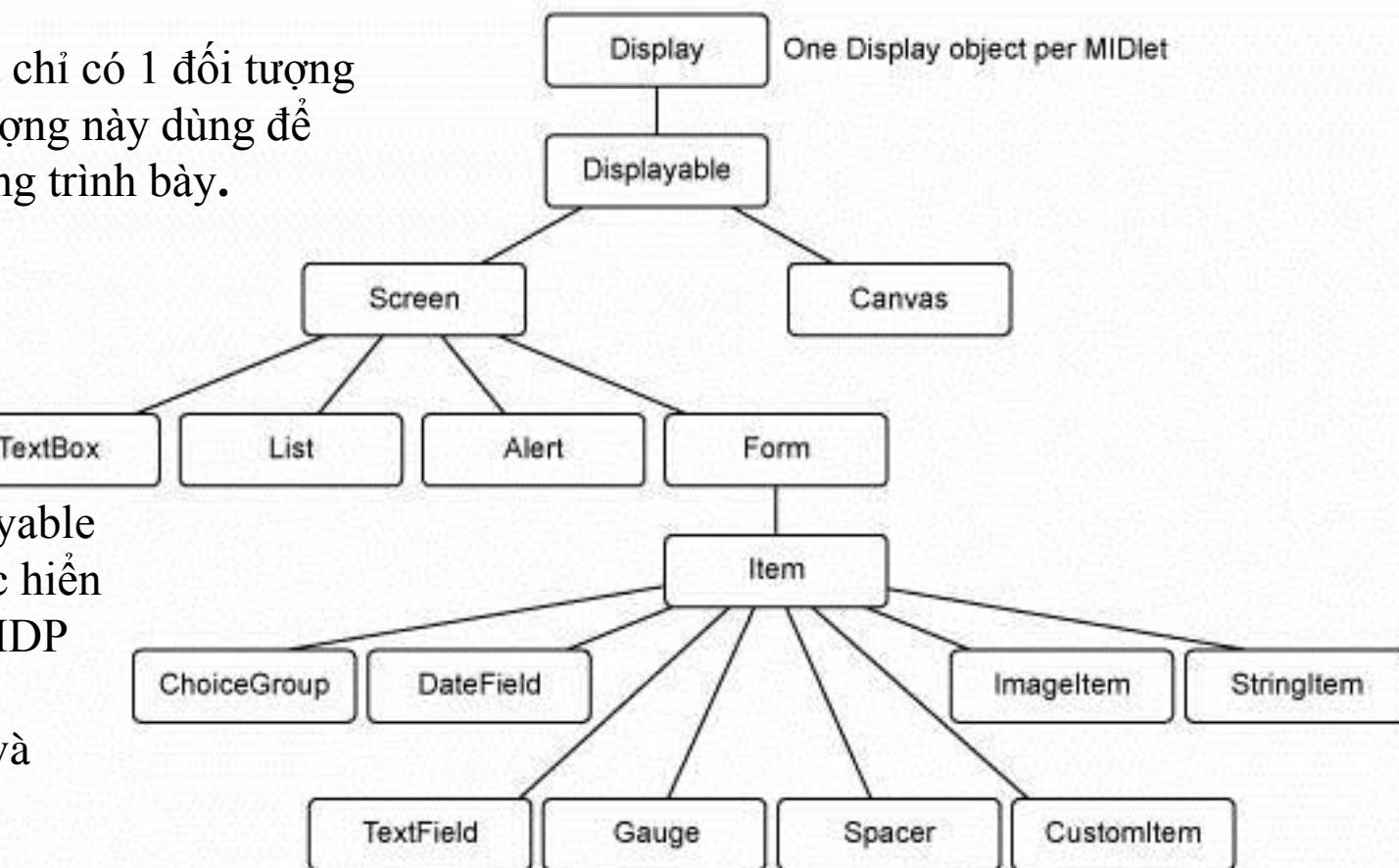
- Classes: MIDlet

Ta có thể khai báo: import javax.microedition.midlet.*;

Hay : import javax.microedition.midlet.MIDlet;

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

- 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.



- 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**.

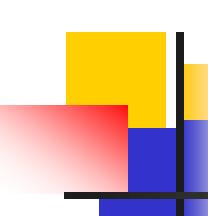
- Một đối tượng **Screen** không phải là một cái gì hiện ra trên thiết bị, lớp **Screen** sẽ được thừa kế bởi các thành phần hiển thị ở mức cao, chính các thành phần này sẽ được hiển thị ra trên màn hình.

Tạo Form

- Tạo Form :Form(String title, Item[] items);
- Ví dụ: **TaoForm**

```
import javax.microedition.lcdui.Display;  
import javax.microedition.lcdui.Form;  
import javax.microedition.lcdui.StringItem;  
import javax.microedition.lcdui.TextField;  
import javax.microedition.midlet.MIDlet;  
public class CreateForm extends MIDlet {  
    protected Display display;  
    // Flag indicating first call of startApp  
    protected boolean started;  
    protected void startApp() {  
        if (!started) {  
            display = Display.getDisplay(this);  
            Form form = new Form("Tieu de Form");  
            form.append("Chao");
```

```
            form.append("Tat ca cac ban");  
            form.append("\nChung ta bat dau lam viec nao!\n  
Mot dong moi\n");  
            form.append("Day la mot dong rat dai chung ta  
khong viet chung tren mot dong duoc");  
            form.append(new TextField("Ho va ten:", "Le Thi  
Cham Chi", 32, TextField.ANY));  
            form.append("Dia chi:");  
            form.append(new TextField(null, null, 32,  
                TextField.ANY));  
            display.setCurrent(form);  
            started = true;  
        }  
    }  
    protected void pauseApp() {}  
    protected void destroyApp(boolean unconditional)  
    {}  
}
```



Các hành động

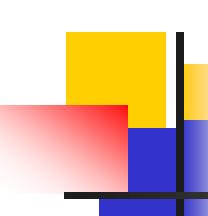
- Các hành động (Command) như: Thoát (EXIT), trở lại (BACK) và gọi một phương thức ..Các Type: BACK, EXIT,CANCEL,HELP,ITEM,SCREEN,STOP,OK

Command(String label, int commandType, int priority);

Ví dụ:

```
private Command cmExit; // khai báo
cmExit = new Command("Exit", Command.EXIT, 1); // tạo hành động thoát
fmMain.addCommand(cmExit); // đưa hành động vào Form
fmMain.setCommandListener(this); // Listen for Event
public void commandAction(Command c, Displayable s) { // Thực thi nó
    if (c == cmExit){
        destroyApp(false);
        notifyDestroyed();
    }
}
```

- Ví dụ: **CacHanhDong**



Thành phần Form

- Khai báo: import javax.microedition.lcdui.Screen
- Một Form chỉ đơn giản là một khung chứa các thành phần, mà mỗi thành phần được thừa kế từ lớp Item
 - StringItem
 - TextField
 - DateField
 - Gauge
 - ChoiceGroup
 - Image and ImageItem
 - CustomItem
 - Spacer

StringItem và TextField

- 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)`
- Ví dụ StringItem :**chuoi, chuoi2, chuoi3**
- Một thành phần **TextField** như bất kỳ các đối tượng nhập văn bản tiêu biểu nào. Có thể chỉ định một nhãn, số ký tự tối đa được phép nhập, và loại dữ liệu được phép nhập. Và TextField còn cho phép nhập vào mật khẩu mà các ký tự nhập vào được che bởi các ký tự mặt nạ. Phương thức dựng của lớp TextField
`TextField(String label, String text, int maxSize, int constraints)`
constraints: 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**: nhập bất kỳ ký tự nào
 - **EMAILADDR**: chỉ cho phép nhập 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
- Ví dụ TextField : **ONhapLieu, TextField1,Login,**

DateField , Gauge

- Thành phần **DateField**: thao tác đối tượng Date, định nghĩa trong java.util.Date. Tạo một đối tượng DateField: chỉnh sửa ngày, giờ hay cả hai. Các phương thức 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:hien thoi,thay doi: **ThoiGian**,
- Thành phần **Gauge**: mô tả mức độ hoàn thành một công việc. Có 2 loại Gauge là loại tương tác(thay đổi Gauge) và không tương tác(cập nhật Gauge).Hàm dựng của lớp Gauge:
Gauge(String label, boolean interactive, int maxValue, int initialValue)
- private Gauge gaVolume; // Điều chỉnh âm lượng
gaVolume = new Gauge("Sound Level", true, 100, 4);
- Ví dụ: **HoanThanh** ; Tổng hợp cả hai: **bai4** hoặc **ktra4**

ChoiceGroup

- Thành phần **ChoiceGroup**: chọn từ một danh sách đầu vào đã được định nghĩa trước.
- **ChoiceGroup(String label, int choiceType, String[] stringElements, Image[] imageElements);**
- ChoiceType có 2 loại:
 - **EXCLUSIVE** (chọn một mục): nhóm này liên quan đến các radio button
 - **MULTIPLE** (chọn nhiều mục): nhóm này liên quan nhóm các checkbox

Ví dụ: chon nhieu muc (CheckBox): **NhomChon**

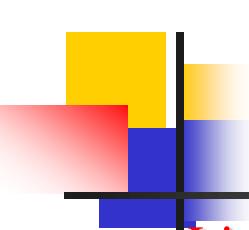
Chon mot muc (Radio): **NhomChonRadio, NhomChonRadio1**

Tổng hợp: **ktra5** (multile); **ktra6** (exclusive)

```
private ChoiceGroup radio1;
private int defaultIndex;
private int RadioGroup;
radio1 = new ChoiceGroup("Moi ban chon:",
                        Choice.EXCLUSIVE);
radio1.append("Chon 1", null);
radio1.append("Chon 2", null);
defaultIndex = radio1.append("Chon 3", null);
radio1.setSelectedIndex(defaultIndex, true);
radioButtonsIndex = form.append(radio1);
public void itemStateChanged(Item item){
    if (item == radio1){
        StringItem msg = new StringItem("Ban da chon: ",
                                         radio1.getString(radio1.getSelectedIndex()));
        form.append(msg);
    }
}
```

Image and ImageItem

- Hai lớp hiển thị hình ảnh: **Image** và **ImageItem**. Image dùng tạo một đối tượng hình ảnh và giữ thông tin chiều cao, chiều rộng, và dù ảnh có biến đổi hay không. Lớp ImageItem: tấm ảnh sẽ được hiển thị, ví dụ tấm ảnh đặt ở trung tâm, bên trái, bên trên của màn hình.
MIDP đưa ra 2 loại hình ảnh là loại không biến đổi và biến đổi. Một ảnh không biến đổi kể từ lúc nó được tạo ra. Loại ảnh này được đọc từ một tập tin. Một ảnh biến đổi 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. Các phương thức dựng cho lớp Image và ImageItem
 - Image createImage(String name)
 - Image createImage(Image source)
 - Image createImage(int width, int height)
 - Image createImage(Image image, int x, int y, int width,int height, int transform)(TOP|LEFT)
 - ImageItem(String label, Image img, int layout, String altText)
- Form fmMain = new Form("Images");
// Tao mot image
Image img = Image.createImage("/terrain1.png");
// Them vao form
fmMain.append(new ImageItem(null, img, ImageItem.LAYOUT_CENTER, null));
- Ví dụ: **HinhAnh**



Thành phần List, Textbox

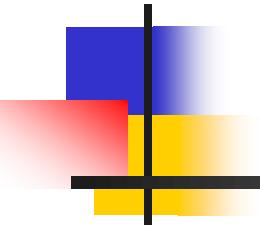
- **List** không tường minh được dùng để thể hiện một thực đơn các chọn lựa.
List(String title, int listType, String[] stringElements, Image[] imageElements);
- Ví dụ: Danh sách cả phần image - **DanhSach**
 - Danh sách chọn kiểu checkbox - **DanhSachCheckBox**
 - chọn các mode (listType) của danh sách - **DanhSach1**
- **TextBox** dùng để cho phép nhập nhiều dòng. TextBox và TextField có ràng buộc giống nhau cho phép nhập liệu. Ví dụ ANY, EMAIL, URI... Phương thức dựng của một TextBox:
TextBox(String title, String text, int maxSize, int constraints)
- Ví dụ: Viết ra lời chào dùng TextBox – **HelloTextBox**
 - Nhap du lieu - **NhapTextBox**

Alert, và Ticker

- Một Alert đơn giản là một hộp thoại rất nhỏ. Có 2 loại Alert:
 - Modal: hộp thoại thông báo được trình bày đến khi người dùng ấn nút đồng ý
 - Non-modal: hộp thoại thông báo chỉ được trình bày trong một số giây nhất địnhCác phương thức dựng của Alert:
`Alert(String title)`
`Alert(String title, String alertText, Image alertImage, AlertType alertType)`
AlertType sử dụng âm thanh để thông báo cho người dùng biết có một sự kiện xảy ra.
AlertType bao gồm 5 loại âm thanh định sẵn là: thông báo, xác nhận, báo lỗi, thông báo và cảnh báo. Các phương thức dựng của Alert cho biết là Alert có thể bao gồm 1 tham chiếu đến một đối tượng AlertType.
- Ví dụ: Thông báo có sử dụng ảnh – **ThongBao1**
Hai loại thông báo – **ThongBao2**
Các loại thông báo - **HopThoaiBao**
- **Ticker** thể hiện một đoạn chuỗi chạy theo chiều ngang. Tham số duy nhất của Ticker là đoạn văn bản được trình bày. Tốc độ và chiều cuộn được xác định bởi việc cài đặt trên thiết bị nào. Phương thức dựng của Ticker
`Ticker(String str)`
Từ cây phân cấp, ta thấy Ticker không là lớp con của lớp Screen mà Ticker là một biến của lớp Screen. Nghĩa là một Ticker có thể được gắn vào bất cứ lớp con của lớp Screen bao gồm cả Alert.
- Ví dụ: Chạy dòng chữ - **ChuoiChay,Ticker1,**

LẬP TRÌNH J2ME CHO THIẾT BỊ DI ĐỘNG

PHẦN 2



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

- Các hàm API cấp cao cho ta tạo ra giao diện các ứng dụng theo chuẩn, các hàm API cấp thấp cho ta thể hiện các ý tưởng của mình. Canvas và Graphics là 2 lớp chính của các hàm API cấp thấp. Bạn làm tất cả các công việc bằng tay. Canvas là một khung vẽ mà người phát triển vẽ lên thiết bị trình bày và xử lý sự kiện. Lớp Graphics cung cấp các công cụ vẽ như drawRoundRect() và drawString()
- **Lớp Canvas:**cung cấp một khung vẽ tạo giao diện tùy biến người dùng. Đa số các phương thức trong lớp này 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

Hệ thống trục tọa độ, tạo một đối tượng Canvas

- Hệ tọa độ cho lớp Canvas: tâm tọa độ là điểm trái, trên của thiết bị. Trị x tăng dần về phải, trị y tăng dần khi xuống dưới. 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
- Đầu tiên tạo ra một lớp thừa kế từ lớp Canvas
class TestCanvas extends Canvas implements CommandListener{
 private Command cmdExit;

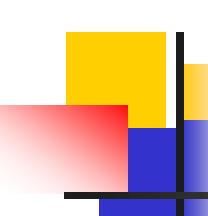
 display = Display.getDisplay(this);
 cmdExit = new Command("Exit", Command.EXIT, 1);
 addCommand(cmdExit);
 setCommandListener(this);

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

TestCanvas canvas = new TestCanvas(this);
- Phương thức paint của lớp Canvas cho phép bạn vẽ các hình dạng, vẽ ảnh, xuất chuỗi

Sự kiện hành động

- Một Canvass có thể xử lý các Command. Chúng ta có thể xử lý các sự kiện Command trên thành phần Canvas cung cách như các thành phần khác
- Mã phím
Trường hợp xử lý các hành động của các phím mềm, một Canvass có thể truy cập đến 12 mã phím. Những mã này được đảm bảo 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_POUNDNăm phương thức để xử lý các mã phím là:
`void keyPressed(int keyCode);`
`void keyReleased(int keyCode);`
`void keyRepeat(int keyCode);`
`String getKeyname(int keyCode);`
- Ví dụ sau: Xử lý các phím, viết ra mã phím – **KeyEvents**,
viết ra dung ham getKeyName() - **KeyCodes**



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

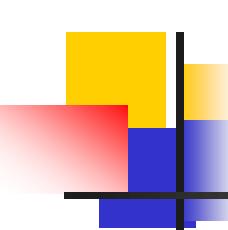
Đơn giản thì các giá trị này được ánh xạ thành các phím mũi tên chỉ hướng của thiết bị, nhưng không phải tất cả các thiết bị di động đều có những giá trị này. Nếu một thiết bị di động thiếu các phím mũi tên thì các hành động của trò chơi sẽ được ánh xạ vào các nút bấm, ví dụ phím trái được ánh xạ vào phím số 2, phím phải được ánh xạ vào phím số 5, và cứ tiếp tục như thế.

Xác định các hành động của trò chơi

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

```
protected void keyPressed(int keyCode) {  
    int gameAction = getGameAction(keyCode);  
    switch(gameAction) {  
        case UP: mMessage = "UP"; break;  
        case DOWN: mMessage = "DOWN"; break;  
        case LEFT: mMessage = "LEFT"; break;  
        case RIGHT: mMessage = "RIGHT"; break;  
        case FIRE: mMessage = "FIRE"; break;  
        case GAME_A: mMessage = "GAME_A"; break;  
        case GAME_B: mMessage = "GAME_B"; break;  
        case GAME_C: mMessage = "GAME_C"; break;  
        case GAME_D: mMessage = "GAME_D"; break;  
        default: mMessage = ""; break;  
    } }  
}
```

- Ví dụ: hiển thị các phím xử lý sự kiện - **KeyMIDlet**



Lớp Graphics

- Chúng ta sử dụng đối tượng Graphics để vẽ lên một Canvas.
- boolean `isColor();` // thiết bị có hỗ trợ hiển thị màu không?
- int `numColors();` // gọi để xác định số màu

Mặc định (DefaultColorPhone) là 4096 colors (0x1000)

`isColor=true; colorCount=0x1000;`

- Các phương thức lấy về màu và thiết lập màu:

`void setColor(int RGB);` //Đặt màu hiện thời qua giá trị RGB

`void setColor(int red, int green, int blue);` // đặt màu, các giá trị red,green.. từ 0-255

`int getColor();` // trả về màu hiện thời

`int getBlueComponent();` // trả về thành phần màu xanh da trời của màu hiện thời 0-255

`int getGreenComponent();` // trả về thành phần màu lục của màu hiện thời 0-255

`int getRedComponent();` // trả về thành phần màu đỏ của màu hiện thời 0-255

`void setGrayScale(int value);` // đặt mức xám

`int getGrayScale();` // trả về giá trị xám từ 0-255

Ví dụ: BLACK = 0; WHITE = 0xffffffff; RED = 0xf96868; GREY = 0xc6c6c6;
LT_GREY = 0xe5e3e3;

Hay `int red = 0, green = 128, blue = 255;`

Sau đó đặt màu: `g.setColor(WHITE); hoặc g.setColor(red, green, blue);`

Vẽ cung và hình chữ nhật

- Chọn nét khi vẽ đường thẳng, cung và hình chữ nhật trên thiết bị hiển thị.

```
int getStrokeStyle(); // trả về kiểu nét vẽ  
void setStrokeStyle(int style); // đặt kiểu nét vẽ  
Hai kiểu nét vẽ được định nghĩa trong lớp Graphics là nét chấm, và nét liền  
g.setStrokeStyle(Graphics.DOTTED);  
g.setStrokeStyle(Graphics.SOLID);
```

- 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);  
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.

Ví dụ: **VeCungCanvas**

- 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);
```

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).

Ví dụ: **VeHinhChuNhat**

Font chữ

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_PROPORIONAL
```

```
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

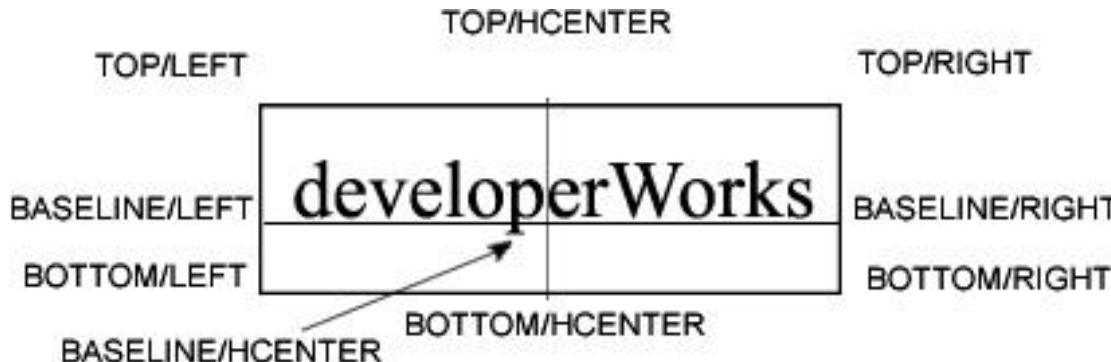
```
Font font = Font.getFont(Font.FACE_PROPORIONAL,  
Font.STYLE_BOLD | Font.STYLE_ITALIC, Font.SIZE_MEDIUM);
```

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 tính của nó.

Ví dụ: **FontChuDonGian**

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



Chiều ngang

LEFT (Bên trái)

HCENTER (Chính giữa của chiều ngang)

RIGHT (Bên phải)

Chiều dọc

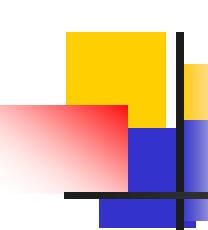
TOP (Ở trên)

BASELINE (Đường thẳng cơ sở)

BOTTOM (Ở dưới)

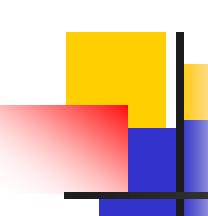
Sử dụng điểm neo phải chỉ ra tọa độ x, y của hình chữ nhật bao quanh.

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



Vẽ các chuỗi ký tự

- 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);
- 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);
}
- Ví dụ: thay đổi Font chữ – **FontChu1**
Dịch chuyên đối tượng – **Ve2**



Các lớp Game trong MIDP 2

■ Với MIDP 2 thì có thêm 5 lớp:

- GameCanvas
- Sprite
- Layer
- LayerManager
- TiledLayer

- Các lớp này tìm trong gói: `java.microedition.lcdui.game`
- Chúng ta không dùng trực tiếp lớp Layer, vì thực chất đây là lớp trừu tượng dùng trong các lớp Spite, LayerManager và TiledLayer.
- Lớp GameCanvas: không phải đợi để thực hiện `keyPressed()` vì ta có phương thức `getKeysState()`. Có kỹ thuật gọi là “*double buffering*” bạn có thể thực hiện vẽ ở “off-screen buffer” khi đó việc copy từ buffer tới các canvas nhanh hơn nhiều. Có thể sử dụng Graphic từ gọi hàm `getGraphics()`, `flushGraphics()` để giải phóng Graphics...
- Ví dụ ta có việc di chuyển chữ ‘x’ dùng GameCanvas – **ExampleGameCanvas**

Lớp Sprite

- Trong Game thì đồ họa làm nên thành công rất lớn. Hầu hết các đối tượng đồ họa được phân loại như các dạng đặc biệt của đồ họa gọi là các Sprite. Một Sprite có thể là bullet, monster, enemies, keys và doors và một vài cái gì đó...
- Các Sprite được nhân nhiều lên là các graphic động, các graphic động này được tạo nên từ cùng một Sprite nhưng nhìn từ các góc khác nhau. Đây là một bộ Sprite:



Sprite Constructor

Có 3 hàm khởi tạo với lớp Sprite

`Sprite (Image image);` // Tạo ra khung Sprite đơn, không động

`Sprite (Sprite sprite);` //Tạo ra Sprite mới từ một Sprite

`Sprite (Image image,int frameWidth, int frameHeight);` //Tạo ra Sprite động với từ 2 frame trở lên, frameWidth và frameHeight là độ rộng và chiều cao của 1 Sprite



- Ta có tổng độ rộng là 160 pixels, độ rộng của 1 frame là 32 pixels, chiều cao là 32pixels. Ta có frameWidth và frameHeight là giống nhau cho 1 bộ Sprite (các Sprite khác thì khác nhau).
- Các Graphic thì bao gồm các Sprite mà độ rộng và chiều cao là hằng số, vì số các pixel thì liên quan đến số màu: nếu 1pixel là 8-bit, 16-bit, 24-bit... thì $2^8=256$, $2^{16}=65536...$ màu

Sprite Collision, Display Sprite Sequence

■ Sprite Collision:

collidesWith(Image image, int x, int y, boolean pixelLevel);

collidesWith(Sprite sprite, boolean pixelLevel);

collidesWidth(TiledLayer tiledLayer, Boolean pixelLevel);

Kiểm tra độ rộng qua vị trí góc trái trên của Sprite

■ Sprite Sequence:

getFrameSequenceLength(); // số phần tử trong 1 dãy

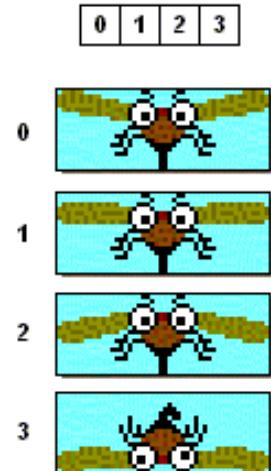
getFrame(); // chỉ số hiện thời của frame trong dãy

nextFrame(); // chỉ tới frame tiếp theo

prevFrame(); // chỉ tới frame trước

setFrame(int sequenceIndex); // đặt frame hiện thời

Default Frame Sequence

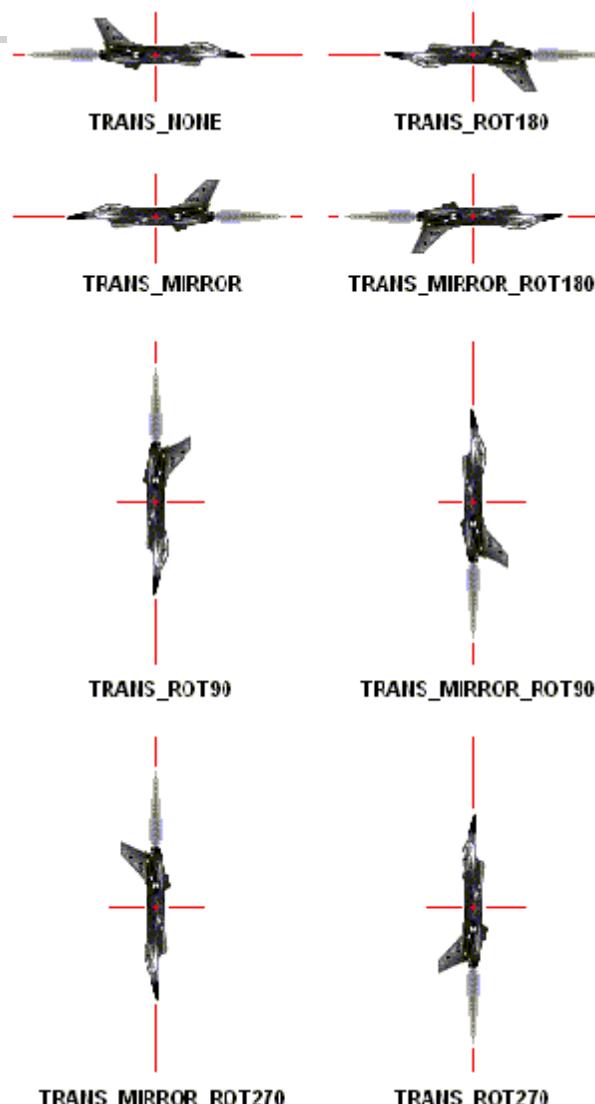


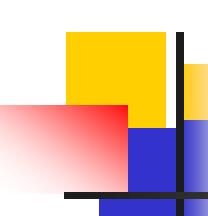
Sprite Transforms

- **Sprite Transforms:** biến dạng có thể tạo Sprite như quay, lấy đối xứng
setTransform(int transform);

TRANS_NONE	0
TRANS_MIRROR_ROT180	1
TRANS_MIRROR	2
TRANS_ROT180	3
TRANS_MIRROR_ROT270	4
TRANS_ROT90	5
TRANS_ROT270	6
TRANS_MIRROR_ROT90	7

- Ví dụ: **ExampleGameSprite**





Lớp LayerManager

- LayerManager để quản lý các lớp Graphic, thứ tự của các lớp giống như chiều thứ 3 (trục z). Thứ tự 0 là lớp gần người dùng nhất.

- Thêm 1 lớp : append(Layer layer);

```
private Sprite playerSprite;
```

```
private Sprite backgroundSprite;
```

```
private LayerManager layerManager;
```

```
Image playerImage = Image.createImage("/transparent.png");
```

```
playerSprite = new Sprite (playerImage,32,32);
```

```
Image backgroundImage = Image.createImage("/background.png");
```

```
backgroundSprite = new Sprite(backgroundImage);
```

```
layerManager = new LayerManager();
```

```
layerManager.append(playerSprite);
```

```
layerManager.append(backgroundSprite);
```

- remove(Layer layer); // loại bỏ một lớp

- insert(Layer layer, int index); // thêm 1 lớp

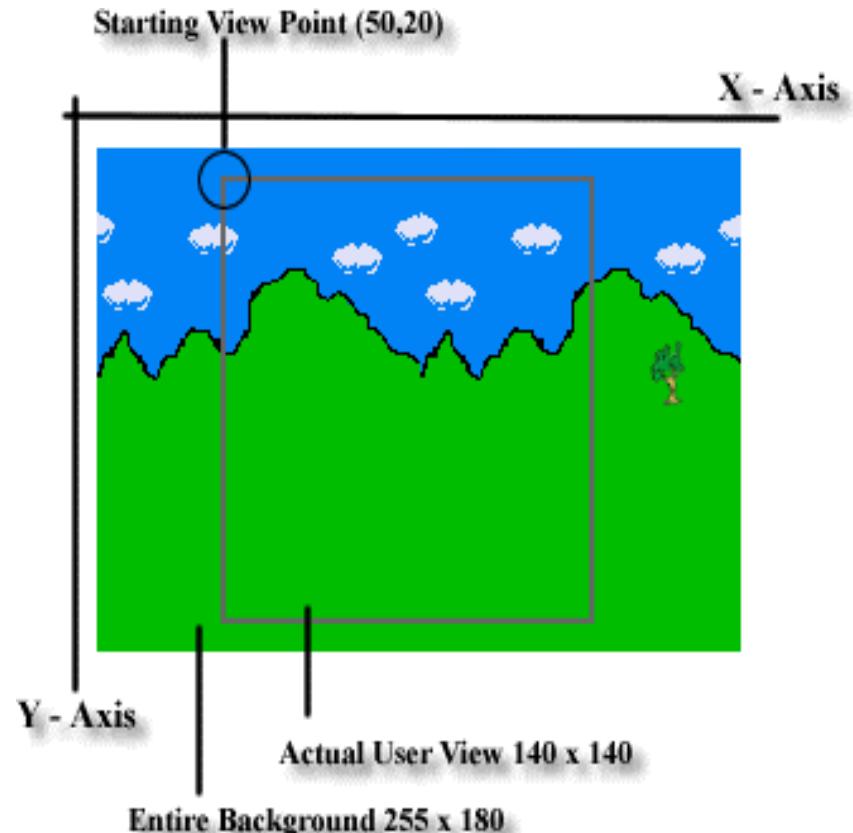
- paint(Graphics g, int x, int y); //muốn hiển thị 1 lớp

Ví dụ : **ExampleLayerManager**

LayerManager and Scrolling Background

- Đôi lúc màn hình nền lớn hơn màn hình hiển thị, lúc đó chúng ta phải cuộn màn hình. Có thể cuộn sang trái, sang phải, lên trên xuống dưới.
- Điểm bắt đầu là (50,20), ta có thể tạo ra điểm động để nó cuộn trên màn hình.

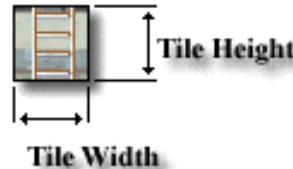
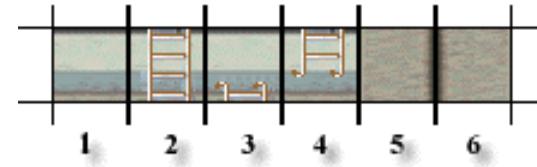
```
if ((keyStates & LEFT_PRESSED) != 0) {  
    if (scnX - 1 > 0)  
        scnX--;  
    }  
    if ((keyStates & RIGHT_PRESSED) != 0) {  
        if (scnX + 1 + 140 < backgroundImage.getWidth())  
            scnX++;  
    }  
    if ((keyStates & UP_PRESSED) != 0){  
        if (scnY-1>0)  
            scnY--;  
    }  
    if ((keyStates & DOWN_PRESSED)! = 0){  
        if (scnY+1+140<backgroundImage.getHeight())  
            scnY++;  
    }
```



Ví dụ: **CuonManHinhNen**

Lớp TiledLayer

- Một TiledLayer là một lưới các ô chia ra từ 1 ảnh.
- Ví dụ hình bên được chia thành 6 vùng, ta chỉ ra các Tile có 32x32 pixel. Tạo nên 1 lớp TiledLayer, mỗi 1 tile này được đánh số (bắt đầu từ 1). Đánh số từ trái sang phải rồi từ trên xuống dưới,
- `TiledLayer(int columns, int rows, Image image, int tileSize, int tileHeight);`
Có số hàng, cột và ảnh cần chia. Độ rộng và cao của tile
- `setCell(int col, int row, int tileIndex);`//đặt tile vào bức ảnh ở vị trí col,row và lấy ảnh có tileIndex (ở trên là từ 1,2,...6)
- `getCell(int col, int row);`//trả về index của cell, nếu cell là empty trả về 0
- `getCellHeight();`//trả về chiều cao của một cell (pixel)
- `getCellWidth();`
- `getColumns();`//trả về số cột của TileLayer
- `getRows();`
- Giống như các Game khác, ta cũng gọi trực tiếp hàm `paint()` hay dùng `LayerManager layerManager.paint(g,0,0);`

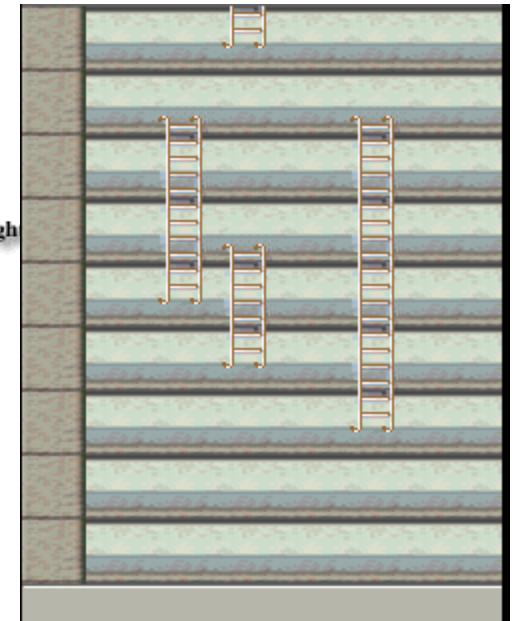
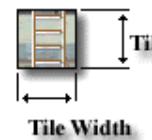


Ví dụ TiledLayer

```
private LayerManager layerManager;  
private TiledLayer tiledBackground;  
tiledBackground = initBackground();  
layerManager = new LayerManager();  
layerManager.append(tiledBackground);
```

```
private TiledLayer initBackground() throws Exception {  
    Image tileImages = Image.createImage("/tiles.png");  
    TiledLayer tiledLayer = new TiledLayer(8,9,tileImages,32,32);
```

```
int[] map = {  
    5, 1, 1, 4, 1, 1, 1, 1,  
    5, 1, 3, 1, 1, 3, 1, 1,  
    5, 1, 2, 1, 1, 2, 1, 1,  
    5, 1, 2, 3, 1, 2, 1, 1,  
    5, 1, 4, 2, 1, 2, 1, 1,  
    5, 1, 1, 4, 1, 2, 1, 1,  
    5, 1, 1, 1, 1, 4, 1, 1,  
    5, 1, 1, 1, 1, 1, 1, 1,  
    5, 1, 1, 1, 1, 1, 1, 1,  
};
```



```
for (int i=0; i < map.length; i++) {  
    int column = i % 8;  
    int row = (i - column) / 8;  
    tiledLayer.setCell(column,row,map[i]);  
}  
return tiledLayer;  
}
```

- Ví dụ : [ExampleTiledLayer](#)

Animated Cells (1)

- Animated Cell là tập hợp động của các Tile tĩnh. Các Animated Tile là các chỉ số âm.

```
int createAnimatedTile(int staticTileIndex);
```

Tạo ra Animated Tile trả về chỉ số âm (-1,-2..)

```
setAnimatedtile(int animatedTileIndex, int staticTileIndex);
```

kết hợp Animated Tile với static tile

- Tạo nền:

```
private TiledLayer initBackground() throws Exception {
```

```
Image tileImages = Image.createImage("/tiles.png");
```

```
TiledLayer tiledLayer = new TiledLayer(8,9,tileImages,32,32);
```

```
int[] map = {
```

```
    5, 1, 1, 4, 1, 1, 1,
```

```
    5, 1, 3, 1, 1, 3, 1, 1,
```

```
    5, 1, 2, 1, 1, 2, 1, 1,
```

```
    5, 1, 2, 3, 1, 2, 1, 1,
```

```
    5, 1, 4, 2, 1, 2, 1, 1,
```

```
    5, 1, 1, 4, 1, 2, 1, 1,
```

```
    5, 1, 1, 1, 1, 4, 1, 1,
```

```
    5, 1, 1, 1, 1, 1, 1, 1,
```

```
    5, 1, 1, 1, 1, 1, 1, 1};
```

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

```
    int column = i % 8;
```

```
    int row = (i - column) / 8;
```

```
    tiledLayer.setCell(column, row, map[i]);
```

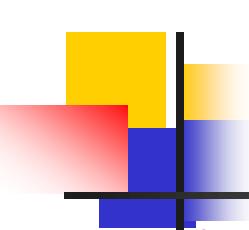
```
}
```

```
animatedIdx = tiledLayer.createAnimatedTile(5);
```

```
tiledLayer.setCell(1, 1, animatedIdx);
```

```
return tiledLayer;
```

```
}
```



Animated Cells (2)

```
public ExampleGameCanvas() throws Exception {  
    super(true);  
    width = getWidth();  
    height = getHeight();  
    currentX = width / 2;  
    currentY = height / 2;  
    delay = 20;  
    tiledBackground = initBackground();  
    layerManager = new LayerManager();  
    layerManager.append(tiledBackground);  
}  
  
public void run() {  
    Graphics g = getGraphics();  
    while (isPlay == true) {  
        input();  
        drawScreen(g);  
        try { Thread.sleep(delay); }  
        catch (InterruptedException ie) {}  
    }  
}
```

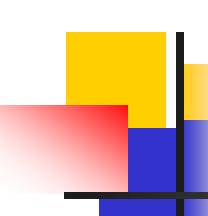
```
private boolean switchTile;  
private int animatedIdx;.....  
if (switchTile) {  
    tiledBackground.setAnimatedTile(animatedIdx,3);  
} else {  
    tiledBackground.setAnimatedTile(animatedIdx,4);  
}  
switchTile = !switchTile;  
layerManager.paint(g,0,0);  
.....  
animatedIdx = tiledLayer.createAnimatedTile(5);  
tiledLayer.setCell(1,1,animatedIdx);
```

Ví dụ: **ExampleTiledLayerAnimated**

Ví dụ : Animation (1)

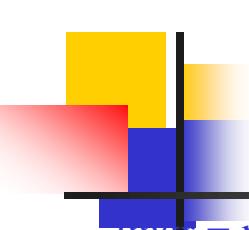
Tạo Sprite tĩnh: AnimationSprite.java

```
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
public class AnimationSprite extends Sprite {
    public AnimationSprite(Image image, int frameWidth, int frameHeight) {
        super(image, frameWidth, frameHeight); } }
    ■ Chạy Animation:
    public void start() {
        running = true;
        Thread t = new Thread(this);
        t.start(); }
    public void run() {
        Graphics g = getGraphics();
        while (running) {
            drawDisplay(g);
            Try {
                Thread.sleep(150); }
            catch (InterruptedException ie) {
                System.out.println("Thread exception"); } } }
    • Draw frame
    private void drawDisplay(Graphics g)
    {
        // Animated sprite, show next frame in sequence
        spSpiral.nextFrame();
        lmgr.paint(g, 0, 0); // Paint layers
        flushGraphics();
    }
```



Ví dụ : Animation (2), AnimationCanvas.java

```
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
public class AnimationCanvas extends GameCanvas implements Runnable {
    private static final int FRAME_WIDTH = 57;
    private static final int FRAME_HEIGHT = 53;
    private AnimationSprite spSpiral; // Animated sprite
    private LayerManager lmgr; // Manage layers
    private boolean running = false; // Thread running?
    public AnimationCanvas() {
        super(true);
        Try {
            spSpiral = new AnimationSprite(Image.createImage("/spiral.png"),
                FRAME_WIDTH, FRAME_HEIGHT);
            spSpiral.defineReferencePixel(FRAME_WIDTH / 2, FRAME_HEIGHT / 2);
            spSpiral.setRefPixelPosition(getWidth() / 2, getHeight() / 2);
```



Ví dụ : Animation (3)

```
lmgr = new LayerManager();
lmgr.append(spSpiral); }
catch (Exception e) {
System.out.println("Unable to read PNG image"); }
}

public void start() {
running = true;
Thread t = new Thread(this);
t.start(); }

public void run() {
Graphics g = getGraphics();
while (running) {
drawDisplay(g);
Try {
Thread.sleep(150); }
catch (InterruptedException ie) {
System.out.println("Thread exception");
} } }
```

```
private void drawDisplay(Graphics g
{
spSpiral.nextFrame();
lmgr.paint(g, 0, 0);
flushGraphics();
}
public void stop()
{
running = false;
}
}
```

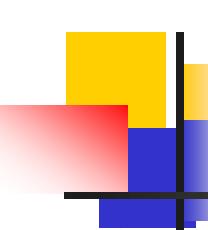
Ví dụ : Animation (4), Animation.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Animation extends MIDlet implements CommandListener {
private Display display; // Reference to display
private AnimationCanvas canvas; // Game canvas
private Command cmExit; // Exit command
public Animation() {
display = Display.getDisplay(this);
cmExit = new Command("Exit", Command.EXIT, 1);
if ((canvas = new AnimationCanvas()) != null) {
canvas.addCommand(cmExit);
canvas.setCommandListener(this); }
public void startApp() {
if (canvas != null) {
display.setCurrent(canvas);
canvas.start(); } }
public void pauseApp() {
public void destroyApp(boolean unconditional) {
canvas.stop(); }
public void commandAction(Command c,
Displayable s) {
if (c == cmExit) {
destroyApp(true);
notifyDestroyed();
} } }
```

Ví dụ: Collisions, AppleSprite.java, CubeSprite.java và StarSprite.java

```
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
public class AppleSprite extends Sprite{
    public AppleSprite(Image image) {
        super(image); // Sprite constructor
        setRefPixelPosition(146, 35); // Set location on canvas
    }
    import javax.microedition.lcdui.game.*;
    import javax.microedition.lcdui.*;
    public class StarSprite extends Sprite {
        public StarSprite(Image image) {
            super(image); // Sprite constructor
            setRefPixelPosition(5, 65); // Set location on canvas
        }
    }
}
```

```
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
public class CubeSprite extends Sprite{
    public CubeSprite(Image image){
        super(image); // Sprite constructor
        // Set location on canvas
        setRefPixelPosition(120, 116);}
}
```



Ví dụ: Collisions, AnimatedSprite.java

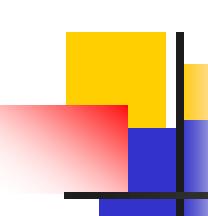
```
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
public class AnimatedSprite extends Sprite {
    public AnimatedSprite(Image image, int frameWidth, int frameHeight) {
        super(image, frameWidth, frameHeight); // Call sprite constructor
    }
}
```

- **Move Sprite:** MoveLeft(), moveRight(int w), moveUp(), moveDown(int h)
- **Khi muốn Move Sprite, ta phải lưu lại vị trí hiện thời saveXY() trong trường hợp Sprite đụng độ với Sprite khác, sau đó trả về vị trí trước đó restoreXY()**

```
private void saveXY() { // Save last position
    previous_x = x;
    previous_y = y;
}
public void restoreXY() {
    x = previous_x;
    y = previous_y;
    setPosition(x, y);
}
```

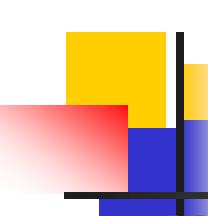
Ví dụ: Collisions, Di chuyển Sprite:ManSprite.java (1)

```
import javax.microedition.lcdui.game.*;
import javax.microedition.lcdui.*;
public class ManSprite extends Sprite{
private int x = 0, y = 0, // Current x/y
previous_x, previous_y; // Last x/y
private static final int MAN_WIDTH = 25; // Width in pixels
private static final int MAN_HEIGHT = 25; // Height in pixels
public ManSprite(Image image){
// Call sprite constructor
super(image);}
public void moveLeft() {
if (x > 0) { // If the man will not hit the left edge...
saveXY();
// If less than 3 from left, set to zero,
// otherwise, subtract 3 from current location
x = (x < 3 ? 0 : x - 3);
setPosition(x, y);
}}
```



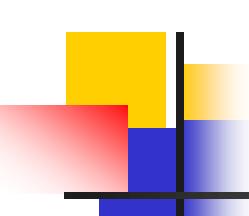
Ví dụ: Collisions, Di chuyển Sprite: ManSprite.java (2)

```
public void moveRight(int w) {  
    if ((x + MAN_WIDTH) < w) { // If the man will not hit the right edge...  
        saveXY();  
        // If current x plus width of ball goes over right side,  
        // set to rightmost position. Otherwise add 3 to current location.  
        x = ((x + MAN_WIDTH > w) ? (w - MAN_WIDTH) : x + 3);  
        setPosition(x, y); } }  
  
public void moveUp() {  
    if (y > 0) { // If the man will not hit the top edge...  
        saveXY();  
        // If less than 3 from top, set to zero,  
        // otherwise, subtract 3 from current location.  
        y = (y < 3 ? 0 : y - 3);  
        setPosition(x, y); } }
```



Ví dụ: Collisions, Di chuyển Sprite: ManSprite.java (3)

```
public void moveDown(int h){  
    if ((y + MAN_HEIGHT) < h) { // If the man will not hit the bottom edge...  
        saveXY();  
        // If current y plus height of ball goes past bottom edge,  
        // set to bottommost position. Otherwise add 3 to current location.  
        y = ((y + MAN_WIDTH > h) ? (h - MAN_WIDTH) : y + 3);  
        setPosition(x, y); } }  
private void saveXY() { // Save last position  
    previous_x = x;  
    previous_y = y; }  
public void restoreXY() {  
    x = previous_x;  
    y = previous_y;  
    setPosition(x, y);  
}  
}
```



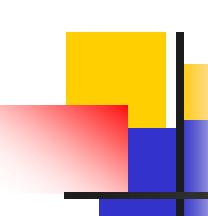
Ví dụ: Collisions, CollisionCanvas.java (1)

- Trả về phím ấn:

```
private void checkForKeys() {  
    int keyState = getKeyStates();  
    if ((keyState & LEFT_PRESSED) != 0) {  
        spMan.moveLeft();  
    }  
    else if ((keyState & RIGHT_PRESSED) != 0) {  
        spMan.moveRight(canvas_width);  
    }  
    else if ((keyState & UP_PRESSED) != 0) {  
        spMan.moveUp();  
    }  
    else if ((keyState & DOWN_PRESSED) != 0) {  
        spMan.moveDown(canvas_height);  
    }  
}
```

- Trả về true nếu có đụng độ

```
private boolean checkForCollision() {  
    if (spMan.collidesWith(spSpiral, true) ||  
        spMan.collidesWith(spApple, true) ||  
        spMan.collidesWith(spCube, true) ||  
        spMan.collidesWith(spStar, true)) {  
        // Upon collision, restore the last x/y position  
        spMan.restoreXY();  
        return true;  
    }  
    else  
        return false;  
}
```



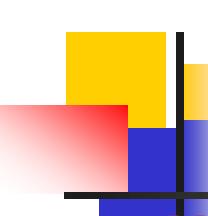
Ví dụ: Collisions, CollisionCanvas.java (2)

■ Lớp CollisionCanvas

```
public class CollisionCanvas extends GameCanvas implements Runnable {  
    private AnimatedSprite spSpiral; // Animated sprite  
    private static final int FRAME_WIDTH = 57; // Width of 1 frame  
    private static final int FRAME_HEIGHT = 53; // Height of 1 frame  
    private int canvas_width, canvas_height; // Save canvas info  
    private ManSprite spMan; // Man (moveable)  
    private AppleSprite spApple; // Apple (stationary)  
    private CubeSprite spCube; // Cube "  
    private StarSprite spStar; // Star "  
    private LayerManager lmgr; // Manage all layers  
    private boolean running = false; // Thread running?  
    private Collisions midlet; // Reference to main midlet
```

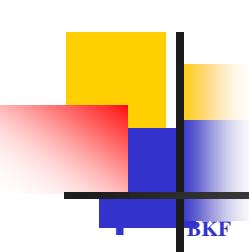
Ví dụ: Collisions, CollisionCanvas.java (3)

```
public CollisionCanvas(Collisions midlet) { // Gamecanvas constructor
    super(true);                                // Create and add to layer
    this.midlet = midlet;                         manager
    Try { // Nonanimated sprites
        spMan = new ManSprite(Image.createImage("/man.png"));
        spApple = new AppleSprite(Image.createImage("/apple.png"));
        spCube = new CubeSprite(Image.createImage("/cube.png"));
        spStar = new StarSprite(Image.createImage("/star.png"));
    } // Animated sprite
    spSpiral = new AnimatedSprite(Image.createImage("/spiral.png"),
        FRAME_WIDTH, FRAME_HEIGHT);
    // Change the reference pixel to the middle of sprite
    spSpiral.defineReferencePixel(FRAME_WIDTH / 2, FRAME_HEIGHT / 2);
    // Center the sprite on the canvas
    // (center of sprite is now in center of display)
    spSpiral.setRefPixelPosition(getWidth() / 2, getHeight() / 2);
    lmgr = new LayerManager();
    lmgr.append(spSpiral);
    lmgr.append(spMan);
    lmgr.append(spApple);
    lmgr.append(spCube);
    lmgr.append(spStar);
```



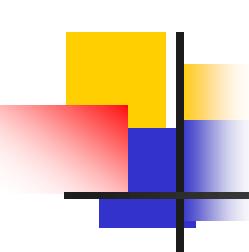
Ví dụ: Collisions, Collisions.java

```
public class Collisions extends MIDlet implements CommandListener {  
    protected Display display; // Reference to display  
    private CollisionCanvas canvas; // Game canvas  
    private Command cmExit; // Exit command  
public Collisions() {  
    display = Display.getDisplay(this);  
    if ((canvas = new CollisionCanvas(this)) != null) { // Create game canvas and exit command  
        cmExit = new Command("Exit", Command.EXIT, 1);  
        canvas.addCommand(cmExit);  
        canvas.setCommandListener(this);  
    }  
}  
public void startApp() {  
    if (canvas != null) {  
        display.setCurrent(canvas);  
        canvas.start();  
    } }  
}
```



PHẦN 3

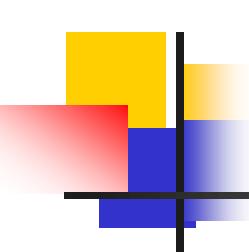
LẬP TRÌNH J2ME CHO THIẾT BỊ DI ĐỘNG



4. PlayerAudio

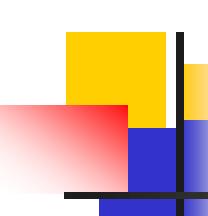
① Giới thiệu

Ngày nay nhờ sự tăng cường hỗ trợ âm thanh trong MIDP2.0, chúng ta có thể tạo những ứng dụng chơi nhạc trên nền Java cho những thiết bị không dây.



②. Lớp Manager (1)

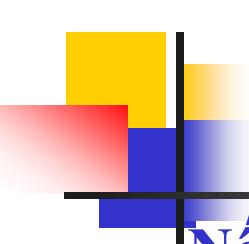
- Manager là điểm truy nhập đặc biệt cho các tài nguyên phụ thuộc hệ thống như là Player cho tiến trình đa phương tiện.
- Manager cung cấp phương thức truy nhập đặc biệt để xây dựng các Player.
- Phương thức: `createPlayer(InputStream stream, String type)` Tạo ra một Player để chơi nhạc từ InputStream.
- Phương thức `createPlayer(String locator)` Tạo ra một Player từ máy dò tìm đầu vào.



②. Lớp Manager (2)

■ Để chơi file nhạc trong máy, chúng ta sử dụng đoạn code như sau:

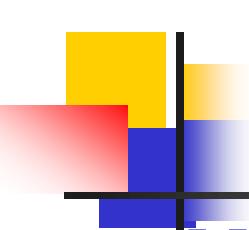
```
try {  
    InputStream is =  
getClass().getResourceAsStream("music.mid");  
    Player p = Manager.createPlayer(is, "audio/midi");  
    p.start();  
} catch (IOException ioe) {}  
} catch (MediaException me) {}  
}
```



②. Lớp Manager (3)

- Nếu muốn chơi file nhạc trên Web Server, làm như sau:

```
try {  
    Player p =  
Manager.createPlayer("http://webserver/music.mid");  
    p.start();  
} catch (IOException ioe) {  
} catch (MediaException me) {  
}
```



②. Lớp Manager (4)

- Manager hỗ trợ chơi các loại file nhạc khác nhau. Có những kiểu được MINE đăng ký, cộng với vài kiểu do người dùng định ra mà nói chung tuân theo cú pháp:

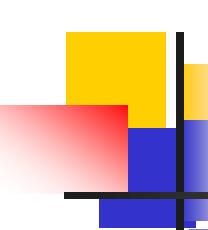
Với file Ware: audio/x-wav

Với file AU: audio/basic

Với file Mp3: audio/mpeg

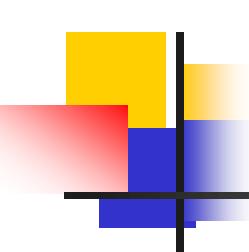
Với file Midi: audio/midi

Với Tone sequences: audio/x-tone-seq



③. Giao diện Player (1)

Player điều khiển quá trình trả lại dữ liệu phương tiện cơ bản. Nó cung cấp các phương thức để quản lý vòng đời của Player, điều khiển tiến trình trả lại và thực thi thành phần trình diễn.



③. Giao diện Player (2)

1. Simple Playback

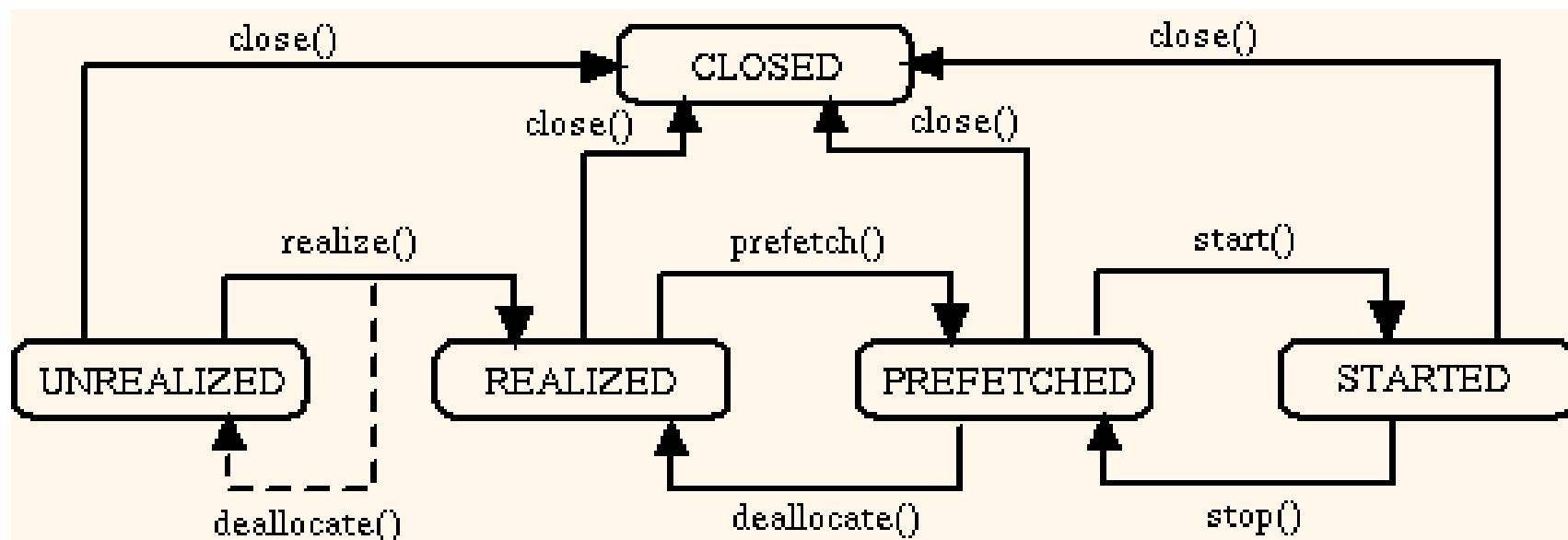
Một Player có thể được tạo ra từ 1 trong các phương thức *Manager's createPlayer*. Sau khi Player được tạo ra, tiến trình gọi sẽ bắt đầu trả lại càng nhanh càng tốt.

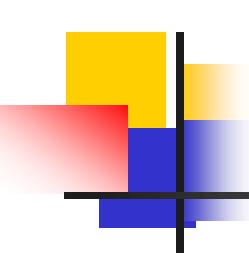
Phương thức sẽ trả lại khi playback được bắt đầu. Việc trả lại sẽ tiếp tục thực hiện ngầm và sẽ tự động kết thúc khi đạt được kết quả.

③. Giao diện Player (3)

2. Vòng đời Player

Player có 5 trạng thái: unrealized, realized, prefetched, started, closed.





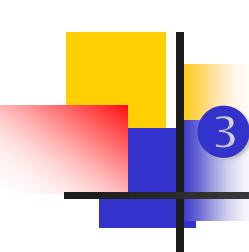
③. Giao diện Player (4)

2. Vòng đời Player

Theo phân loại trên, Player chuyển từ trạng thái **UNREALIZED** sang **REALIZED**, sau đó **PREFETCHED**, cuối cùng **STARTED**.

Player dừng lại khi nó nhận được kết quả cuối cùng của media; hay khi phương thức stop được gọi. Khi việc này xảy ra, Player chuyển từ trạng thái **STARTED** sang **PREFETCHED**. Rồi lặp lại vòng đời.

Để sử dụng Player, ta phải thiết lập tham số để quản lý sự chuyển đổi của nó thông qua các trạng thái và chuyển đổi nó bằng việc sử dụng các phương thức chuyển đổi.



③. Giao diện Player (5)

3. Các trạng thái của Player

3.1. Trạng thái UNREALIZED

Player bắt đầu với trạng thái này. Một Player chưa thực thi không có đủ thông tin để tìm đủ tài nguyên nó cần cho hàm.

Các phương thức không được sử dụng khi ở trạng thái này:

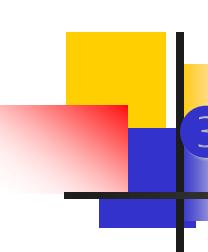
`getContentType`

`setMediaTime`

`getControls`

`getControl`

Phương thức `realize` chuyển Player từ trạng thái UNREALIZED sang REALIZED.

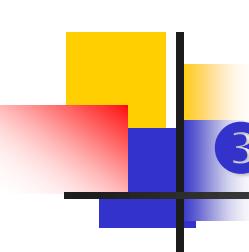


3. Giao diện Player (6)

3.2. Trạng thái REALIZED

Một Player ở trạng thái này khi nó dành được thông tin được yêu cầu cho việc kiểm tài nguyên media. Player đang thực thi có thể là 1 tài nguyên và tiến trình mất nhiều thời gian. Player có thể có giao tiếp với server, đọc file, hay tương tác với 1 tập hợp các đối tượng. Mặc dù realized player không kiểm được tài nguyên nào, nó vẫn có khả năng dành được tất cả tài nguyên mà nó cần trừ những tài nguyên hệ thống khan hiếm, ví dụ: thiết bị audio.

Thông thường, Player chuyển từ trạng thái UNREALIZED sang REALIZED. Sau khi phương thức realize được gọi, chỉ có một cách để trả lại trạng thái UNREALIZED là gọi phương thức deallocate trước khi phương thức realize hoàn thành.



③. Giao diện Player (7)

3.3. Trạng thái PREFETCHED

Ở trạng thái realized, Player vẫn có thể thực thi một số tác vụ mất nhiều thời gian trước khi nó thực sự được bắt đầu.

Một Player ở trạng thái PREFETCHED , nếu nó đã được khởi động. Prefetching làm giảm sự khởi động ngầm của Player đến giá trị nhỏ nhất có thể.

3. Giao diện Player (8)

3.4. Trạng thái STARTED

Player có thể vào trạng thái này bằng cách gọi phương thức start. Một STARTED Player nghĩa là Player đang chạy và đang xử lý dữ liệu. Player trả lại trạng thái PREFETCHED khi nó dừng, khi phương thức stop được gọi.

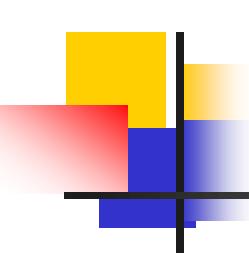
Khi Player chuyển từ trạng thái PREFETCHED sang STARTED, nó cung cấp sự kiện STARTED. Khi nó chuyển từ trạng thái STARTED sang PREFETCHED, nó cung cấp sự kiện STOPPED, END_OF_MEDIA, phụ thuộc vào lý do dừng.

Phương thức không được sử dụng khi nó ở trạng thái này là:

setLoopCount

3.5. Trạng thái CLOSED

Ở trạng thái này Player trả lại hầu như mọi tài nguyên và nó sẽ không được sử dụng lại.

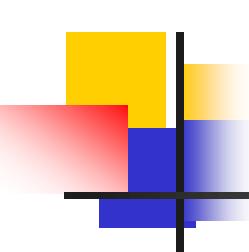


③. Giao diện Player (9)

Chúng ta có thể sử biết được thời gian file
nhạc đã chơi bằng cách sử dụng phung thức
sau:

play.getDuration() / 0xf4240L.

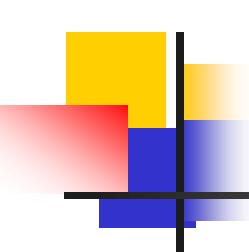
Thời điểm nó dừng là lúc duration của nó đã hết,
hoặc khi nó chuyển từ state STARTED về state
PREFETCHED.



④. Giao diện VolumeControl (1)

VolumeControl là một giao diện để thao tác điều chỉnh âm thanh của một Player.

Giao diện này sẽ cho âm lượng của âm thanh sử dụng một giá trị số nguyên thay đổi từ 0 đến 100(0 là mức thấp nhất, 100 là mức cao nhất).



④. Giao diện VolumeControl (2)

Các phương thức của giao diện này:

getLevel() Lấy âm lượng ở mức hiện tại, giá trị trả về là kiểu int.

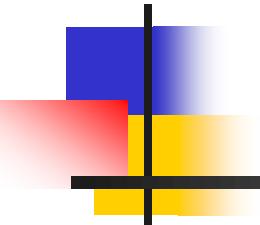
getMuted() Lấy trạng thái mute của tín hiệu liên quan đến VolumeControl này, giá trị trả về là kiểu boolean.

setLevel(int level) Đặt âm lượng sử dụng các giá trị từ 0 đến 100;

setMute(boolean mute) Thiết lập trạng thái mute hoặc unmute.

LẬP TRÌNH J2ME CHO THIẾT BỊ DI ĐỘNG

PHẦN 4



5. Record Management System (RMS)

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

- RMS là hệ thống được tổ chức và quản lý dưới dạng các record (bản ghi). Mỗi bản ghi có thể chứa bất kỳ loại dữ liệu nào: kiểu số nguyên, chuỗi ký tự, một ảnh và kết quả của 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. Kích thước dữ liệu không được vượt quá giới hạn qui định của thiết bị di động. RMS lưu dữ liệu gần như một cơ sở dữ liệu, bao gồm nhiều dòng, mỗi dòng lại có một số định danh duy nhất.
- Một tập các bản ghi(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,sẽ tăng thêm 1 cho các Record tiếp theo. Record ID không là chỉ mục (index), các thao tác xóa Record trong RecordStore sẽ không tính toán lại các Record ID của các Record hiện có cũng không thay đổi Record ID của các Record được tạo mới, ví dụ: xóa record id 3, thêm một record mới sẽ có id là 4. Data là một dãy các byte đại diện cho dữ liệu cần lưu.

Tên được dùng để phân biệt các RecordStore trong bộ các MIDlet (MIDlet suite). MIDlet suite là tập các MIDlet có chung không gian tên (name space), chia sẻ 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 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.

Các vấn đề liên quan đến RMS

- **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.
- **Tốc độ truy xuất dữ liệu :** Các thao tác trên vùng nhớ này sẽ chậm hơn nhiều khi truy xuất dữ liệu trên bộ nhớ RAM. Giống như tốc độ đọc ổ cứng và tốc độ đọc từ RAM của máy tính. Trong kỹ thuật lập trình phải thường xuyên cache dữ liệu và các thao tác liên quan đến RMS chỉ thực hiện tập trung một lần (lúc khởi động hay đóng ứng dụng).
- **Cơ chế luồng an toàn :** Nếu RecordStore chỉ được sử dụng bởi một MIDlet, không phải lo lắng 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

Các hàm API trong RMS (1)

- RecordStore không có hàm khởi tạo.

RecordStore Class: javax.microedition.rms.RecordStore

- *static RecordStore openRecordStore(String recordStoreName, boolean createIfNecessary) :*
Mở một Recordstore, có tham số tạo Record store nếu nó chưa tồn tại.
- Ví dụ: chỉ duy nhất 1 đối tượng RecordStore được tạo mặc dù mở nhiều lần cùng 1 tên

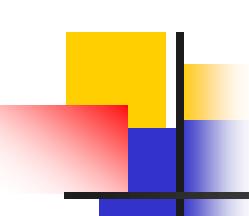
private RecordStore rs = null;

```
static final String REC_STORE = "db_1";
```

```
private void db(String str) {  
    System.err.println("Msg: " + str);  
}
```

```
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.



Các hàm API trong RMS (2)

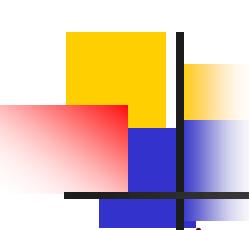
- *void closeRecordStore() : Đóng RecordStore*

- Ví dụ:

```
private RecordStore rs = null;  
public void closeRecStore() {  
    try{  
        rs.closeRecordStore();  
    }  
    catch (Exception e) {  
        db(e.toString());  
    } }
```

- *static void deleteRecordStore(String recordStoreName) : Xóa RecordStore*
- *static String[] listRecordStores() : Danh sách các RecordStore trong MIDlet suite, trả về mảng các chuỗi là tên của RecordStore, nếu không có RecordStore nào thì trả về null*
- Ví dụ:

```
public void deleteRecStore() {  
    if (RecordStore.listRecordStores() != null){  
        try {  
            RecordStore.deleteRecordStore(REC_STORE);  
        }  
        catch (Exception e) {  
            db(e.toString());  
        }  
    } }
```



Các hàm API trong RMS (3)

- *int addRecord(byte[] data, int offset, int numBytes)*: Thêm một record vào RecordStore

- Ví dụ:

```
public void writeRecord(String str) {  
    byte[] rec = str.getBytes();  
    try {  
        rs.addRecord(rec, 0, rec.length);  
    }  
    catch (Exception e) {  
        db(e.toString());  
    } }
```

Trước khi lưu vào RecordStore, cần phải chuyển đổi kiểu string thành dãy byte
`byte[] rec = str.getBytes(); rs.addRecord(rec, 0, rec.length);`

- Có thể thêm một Record rỗng vào RecordStore nếu tham số đầu tiên là *null*. Tham số thứ 2 cho biết vị trí bắt đầu trong mảng các byte và tham số thứ 3 cho biết số byte sẽ được ghi vào RecordStore. Nếu thực hiện thành công, phương thức này trả về số nguyên chỉ số recordID của Record vừa được thêm vào.

Các hàm API trong RMS (4)

- *int getRecord(int recordId, byte[] buffer, int offset)* : Lấy nội dung của record vào dãy byte.
- *int getNumRecords()* : Số lượng các record.
- Ví dụ:

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

Các hàm API trong RMS (5)

- 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:

```
public void readRecords() {  
    try {  
        int len;  
  
        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("-----");  
        }  
    } catch (Exception e) {  
        db(e.toString());  
    }  
}
```

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

```
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(); // tạo record store
        // viết vào record và đọc chúng ra
        writeRecord("J2ME and MIDP");
        writeRecord("Wireless Technology");
        readRecords();
        closeRecStore(); // đóng record store
        deleteRecStore(); // Xoá 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(){ }
```

Ví dụ : đọc và ghi đối tượng string (ReadWrite.java) (2)

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

Ví dụ : đọc và ghi đối tượng string (ReadWrite.java) (3)

```
■ public void readRecords() {  
    try {  
        byte[] recData = new byte[50];  
        int len;  
        for (int i = 1; i <= rs.getNumRecords(); i++)  
        {  
            len = rs.getRecord( i, recData, 0 );  
            System.out.println("Record #" + i + ": " +  
                new String(recData, 0, len));  
            System.out.println("-----");  
        }  
    } catch (Exception e) {  
        db(e.toString());  
    }  
}  
private void db(String str){  
    System.err.println("Msg: " + str);  
}  
}
```

- Thực hiện: **ReadWrite**

Chuyển đổi dữ liệu giữa Record và Mảng các byte (1)

- Dữ liệu được RMS lưu trữ trong các Record là một chuỗi (mảng) các byte. Trong ứng dụng, bạn muốn lưu trữ nhiều kiểu dữ liệu khác nhau: một số nguyên hay là chuỗi các ký tự. Vậy, trước khi lưu dữ liệu vào Record cần phải chuyển dữ liệu này thành mảng các byte. Vấn đề là phải tìm ra các đối tượng trung gian giúp việc lưu trữ dữ liệu vào Record và đọc dữ liệu từ các Record một cách dễ dàng, thuận tiện và hiệu quả. CLDC cung cấp các lớp:
java.io.ByteArrayInputStream, *java.io.ByteArrayOutputStream*, *java.io.DataInputStream*,
java.io.DataOutputStream được thừa hưởng từ J2SE dễ dàng trong việc chuyển đổi dữ liệu qua lại giữa RMS và ứng dụng trong gói *java.io*.
- Lớp *java.io.ByteArrayInputStream* chuyển đổi một mảng các byte thành một luồng vào(input stream), và cung cấp các phương thức thao tác dữ liệu trên nó một cách dễ dàng. Ví dụ đọc và in ra tất cả các giá trị trong mảng các byte.

```
byte[] data = new byte[]{ 1, 2, 3 };

ByteArrayInputStream bin = new ByteArrayInputStream( data );

int b;

while( ( b = bin.read() ) != -1 ){

    System.out.println( b );

}

try {

    bin.close();

} catch( IOException e ){} 
```

Chuyển đổi dữ liệu giữa Record và Mảng các byte (2)

- Lớp *java.io.ByteArrayOutputStream* : ghi dữ liệu kiểu byte lên mảng các byte. Ví dụ ghi vào *ByteArrayOutputStream* ba giá trị 1, 2, 3 liên tiếp sau đó xuất ra mảng các byte.

```
ByteArrayOutputStream bout = new ByteArrayOutputStream();
bout.write( 1 );  bout.write( 2 );  bout.write( 3 );
byte[] data = bout.toByteArray();
for( int i = 0; i < data.length; ++i ){
    System.out.println( data[i] );
}
try {
    bout.close();
} catch( IOException e ){
// bỏ qua các ngoại lệ
}
```

- Bộ đếm (buffer) của *ByteArrayOutputStream* sẽ tự động tăng dần lên khi ghi dữ liệu lên nó, phương thức *toByteArray()* của nó sẽ chép tất cả dữ liệu lên mảng các byte.

Ghi dữ liệu kiểu cơ bản trên Record

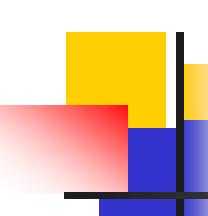
Ghi dữ liệu có các kiểu cơ bản như *int*, *long*, *String* lên một Record thực hiện bằng các lớp *java.io.ByteArrayOutputStream* và *java.io.DataOutputStream*.

```
private RecordStore rs = null;  
public void writeStream(String[] sData, boolean[] bData, int[] iData) {  
    try {  
        // Write data into an internal byte array  
        ByteArrayOutputStream strmBytes = new ByteArrayOutputStream();  
        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]);  
            strmDataType.flush(); // Clear any buffered data  
            record = strmBytes.toByteArray(); // Get stream data into byte array and write record  
            rs.addRecord(record, 0, record.length);  
            strmBytes.reset(); }  
        strmBytes.close();  
        strmDataType.close(); }  
    catch (Exception e) {  
        db(e.toString()); } }
```

Đọc dữ liệu kiểu cơ bản trên Record

- Đọc dữ liệu từ Record ngược lại bằng các lớp *java.io.ByteArrayInputStream* và *java.io.DataInputStream*.

```
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("-----");  
            strmBytes.reset(); // Reset so read starts at beginning of array  
        }  
        strmBytes.close();  
        strmDataType.close();  
    }  
    catch (Exception e){  
        db(e.toString());  
    } }  
}
```



Ghi và đọc sử dụng stream

- 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.
- Khi sử dụng DataOutputStream và DataInputStream, cần phải đọc và ghi theo đúng thứ tự, nếu không sẽ không ra kết quả mong muốn
- Ví dụ: **ReadWriteStreams**

Duyệt Record với 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());
    ...
}
```
- RecordEnumeration Interface: javax.microedition.rms.RecordEnumeration
 - 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 có 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

Sắp xếp các record với interface RecordComparator (1)

- Interface định nghĩa phương thức compare với trị đầu là hai mảng các byte thể hiện hai Record cần so sánh. Phương thức này trả về các trị 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 2Do RecordComparator 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;
 }
}
```

# Sắp xếp các record với interface RecordComparator (2)

- Trong hàm readRecord(), khi tạo Enumeration ta đã tham chiếu đến đối tượng comp của lớp Comparator, khi enumerator tạo index cho RecordStore nó sẽ sử dụng hàm compare() ở trên để sắp xếp các record (record là dạng text - hàm String.CompareTo() )
- ```
public void readRecords() {  
    try {  
        if (rs.getNumRecords() > 0){  
            Comparator comp = new Comparator();  
            RecordEnumeration re = rs.enumerateRecords(null, comp, false);  
            while (re.hasNextElement()) {  
                // Calls String constructor that takes an array of bytes as input  
                String str = new String(re.nextRecord());  
                System.out.println(str);  
                System.out.println("-----");  
            } }  
        catch (Exception e){  
            db(e.toString());  
        } }  
    } Ví dụ: SimpleSort
```

Sắp xếp các record với dữ liệu kiểu cơ bản

- Nếu ghi nhiều kiểu dữ liệu vào trong một record:

```
strmDataType.writeUTF("Text 1");
strmDataType.writeBoolean(true);
strmDataType.writeInt(1);
```
- Các kiểu dữ liệu sẽ lưu vào một stream dạng binary. Các stream này được chuyển thành mảng và đưa vào recordstore: record = strmBytes.toByteArray();

```
rs.addRecord(record, 0, record.length);
```
- Với kiểu dữ liệu binary ta phải viết lại hàm compare() thực hiện chức năng chuyển đổi chuỗi byte và sắp xếp đúng kiểu dữ liệu. Thực thi interface RecordComparator để sắp xếp record chứa nhiều kiểu dữ liệu. Đây là dữ liệu sẽ lưu vào recordstore:

```
String[] names = {"Thu", "Hanh", "Yen", "Khanh", "Anh"};
boolean[] sex = {false, true, false, true, true};
int[] rank = {2, 0, 4, 3, 1};
```
- Khi lưu vào recordstore sẽ có dạng như sau:
Record #1
Name : Anh
Sex : Male
Rank : 1

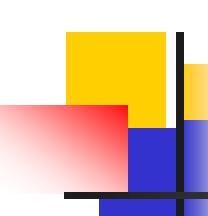
Sắp xếp các record với dữ liệu kiểu String (1)

```
public void readStream() {  
    try {  
        byte[] recData = new byte[50];  
        ByteArrayInputStream strmBytes = new ByteArrayInputStream(recData);  
        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()){  
                rs.getRecord(re.nextRecordId(), recData, 0);  
                System.out.println("Record #" + i++);  
                System.out.println("Name: " + strmDataType.readUTF());  
                if (strmDataType.readBoolean())  
                    System.out.println("Sex: Male");  
                else  
                    System.out.println("Sex: Female" );  
                System.out.println("Rank: " + strmDataType.readInt());  
                System.out.println("-----");  
                strmBytes.reset();}  
                comp.compareStringClose(); re.destroy(); }  
                strmBytes.close(); strmDataType.close(); }  
            catch (Exception e) { db(e.toString()); } }  
        
```

Sắp xếp các record với dữ liệu kiểu String (2)

```
class ComparatorString implements  
RecordComparator {  
private byte[] recData = new byte[10];  
// Read from a specified byte array  
private ByteArrayInputStream strmBytes = null;  
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;  
}}}
```



Sắp xếp các record với dữ liệu kiểu String (3)

- 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  
str1 = strmDataType.readUTF();  
// Read record #2  
...  
str2 = strmDataType.readUTF();  
// Compare record #1 and #2  
int result = str1.compareTo(str2);
```

- Ví dụ: **StringSort**

Sắp xếp các record với kiểu integer, Ví dụ: IntSort

Tiêu chí sắp xếp là theo kiểu integer, dữ liệu ta cần lấy nằm cuối cùng trong dãy byte do đó cần phải đọc theo thứ tự, phải đọc kiểu String, boolean rồi mới đến integer:

```
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 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  
        if (x1 == x2) // Compare record #1 and #2  
            return RecordComparator.EQUIVALENT;  
        else if (x1 < x2)  
            return RecordComparator.PRECEDES;  
        else  
            return RecordComparator.FOLLOWS; }  
    catch (Exception e) {  
        return RecordComparator.EQUIVALENT;  
    } }
```

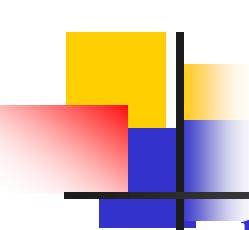
Searching with RecordFilter (1)

- enumerator cung cấp cơ chế lọc (tìm kiếm các record thỏa mãn một điều kiện nào đó). Sử dụng RecordComparator tất cả các record trong RecordStore đều được lưu trong một result set. Dùng RecordFilter, thỏa điều kiện mới trong enumerator result set.

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

- Class RecordFilter được gắn với một enumerator, nó dùng hàm matches() duyệt hết recordstore lấy ra những record cần tìm:

```
SearchFilter search = new SearchFilter("search text"); // Create a new search filter  
// Reference the filter when creating the result set  
RecordEnumeration re = rs.enumerateRecords(search,null,false);  
boolean matches(byte[] candidate) : Tìm kiếm record thỏa mãn một điều kiện nào đó
```



Searching with RecordFilter (2)

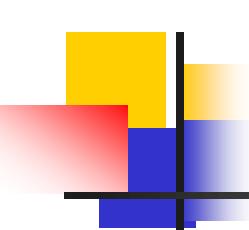
```
■ public SimpleSearch() {  
    display = Display.getDisplay(this);  
    // Define textfield, stringItem and commands  
    tfFind = new TextField("Find", "", 10, TextField.ANY);  
    siMatch = new StringItem(null, null);  
    cmExit = new Command("Exit", Command.EXIT, 1);  
    cmFind = new Command("Find", 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  
}
```

Searching with RecordFilter (3), Ví dụ: SimpleSearch

```
■ 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);  
            if (re.numRecords() > 0) // A match was found using the filter  
                siMatch.setText(new String(re.nextRecord())); // Show match in the stringItem on the form  
            re.destroy(); // Free enumerator  
        } }  
        catch (Exception e) {  
            db(e.toString()); } }  
  
■ class SearchFilter implements RecordFilter {  
    private String searchText = null;  
    public SearchFilter(String searchText) {  
        this.searchText = searchText.toLowerCase(); // This is the text to search for  
    }  
    public boolean matches(byte[] candidate) {  
        String str = new String(candidate).toLowerCase();  
        if (searchText != null && str.indexOf(searchText) != -1) // Look for a match  
            return true;  
        else  
            return false;  
    } }
```

Searching with RecordFilter – SearchStreams (1)

- Đữ liệu lưu vào record dạng dãy byte, trong dãy byte lưu nhiều trường dữ liệu:
`strmDataType.writeUTF(sData); // Write Strings`
`strmDataType.writeBoolean(bData); // Write booleans`
`strmDataType.writeInt(iData); // 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;`



Searching with RecordFilter – SearchStreams (2)

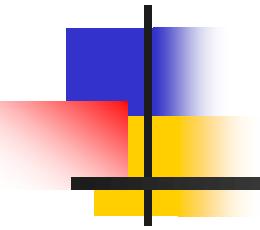
- public void writeTestData() {
String[] names = {"Lan : Lop C04 CNTT HVCNBCVT",
"Thu : K45 CNTT Dai Hoc Bach Khoa HN",
"Hoai Anh : K39 QTDN Truong Kinh Te Quoc Dan",
"Yen Chi : Lop Anh Ngu Truong Dai Hoc Ngoai Ngu HN"};
boolean[] sex = {true, false, true, true};
int[] rank = {3, 0, 1, 2};
writeStream(names, sex, rank); }
■ 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);
if (re.numRecords() > 0) // A match was found using the filter
{ // Read from the specified byte array

Searching with RecordFilter – SearchStreams (3)

```
ByteArrayInputStream strmBytes = new ByteArrayInputStream(re.nextRecord());
DataInputStream strmDataType = new DataInputStream(strmBytes); // Read Java data types from
the above byte array
siMatch.setText(strmDataType.readUTF()); // Show matching result in stringItem component on form
search.searchFilterClose(); // Close record filter
strmBytes.close(); // Close stream
strmDataType.close(); // Close stream
re.destroy(); // Free enumerator
} } }
catch (Exception e) {
db(e.toString());
}
}
```

LẬP TRÌNH J2ME CHO THIẾT BỊ DI ĐỘNG

PHẦN 5



Eliminator: Game Menu, EliminatorBasicMenu (1)

■ Basic Main Menu

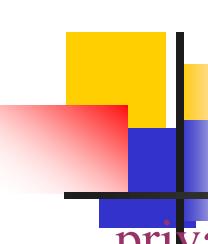
```
import javax.microedition.lcdui.*;
public class MainMenuScreen extends List
implements CommandListener {
private Eliminator midlet;
private Command selectCommand = new
Command("Select", Command.ITEM,1);
private Command exitCommand = new
Command("Exit", Command.EXIT,1);
private Alert alert;
public MainMenuScreen(Eliminator midlet) {
super("Eliminator",Choice.IMPLICIT);
this.midlet = midlet;
append("New Game",null);
append("Settings",null);
append("High Scores", null);
append("Help",null);
append("About",null);
```

```
addCommand(exitCommand);
addCommand(selectCommand);
setCommandListener(this);
}
public void commandAction(Command c,
Displayable d) {
if (c == exitCommand) {
midlet.mainMenuScreenQuit();
return;
} else if (c == selectCommand) {
processMenu(); return;
} else {
processMenu(); return;
}}
```

Eliminator: Game Menu, EliminatorBasicMenu (2)

```
private void processMenu() {  
    try {  
        List down = (List)midlet.display.getCurrent();  
        switch (down.getSelectedIndex()) {  
            case 0: scnNewGame(); break;  
            case 1: scnSettings(); break;  
            case 2: scnHighScores(); break;  
            case 3: scnHelp(); break;  
            case 4: scnAbout(); break;}  
    } catch (Exception ex) {  
        // Proper Error Handling should be done here  
        System.out.println("processMenu::"+ex); } }  
  
private void scnNewGame() {  
    midlet.mainMenuScreenShow(null); }  
  
private void scnSettings() {
```

```
    alert = new  
    Alert("Settings","Settings.....",null,null);  
  
    alert.setTimeout(Alert.FOREVER);  
  
    alert.setType(AlertType.INFO);  
  
    midlet.mainMenuScreenShow(alert);  
}  
  
private void scnHighScores() {  
    alert = new Alert("High Scores"  
    ,"High Scores.....",null,null);  
  
    alert.setTimeout(Alert.FOREVER);  
  
    alert.setType(AlertType.INFO);  
  
    midlet.mainMenuScreenShow(alert);  
}
```



Eliminator: Game Menu, EliminatorBasicMenu (3)

```
private void scnHelp() {  
    alert = new Alert("Help","Help.....",null,null);  
    alert.setTimeout(Alert.FOREVER);  
    alert.setType(AlertType.INFO);  
    midlet.mainMenuScreenShow(alert);  
}  
  
private void scnAbout() {  
    alert = new Alert("About","Eliminator\nVersion 1.0.0\nby Jason Lam",null,null);  
    alert.setTimeout(Alert.FOREVER);  
    alert.setType(AlertType.INFO);  
    midlet.mainMenuScreenShow(alert);  
}  
}
```

Eliminator: Game Menu, EliminatorBasicMenu (4)

Main Midlet Source Code:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class Eliminator extends MIDlet {
protected Display display;
private Image splashLogo;
private boolean isSplash = true;
MainMenuScreen mainMenuScreen;
public Eliminator() {}
public void startApp() {
display = Display.getDisplay(this);
mainMenuScreen = new
MainMenuScreen(this);
if(isSplash) {
```

```
isSplash = false;
try {
splashLogo =Image.createImage("/splash.png");
new SplashScreen(display, mainMenuScreen,
splashLogo,3000);
} catch(Exception ex) {
mainMenuScreenShow(null);
}
} else {
mainMenuScreenShow(null);
}
```

Eliminator: Game Menu, EliminatorBasicMenu (5)

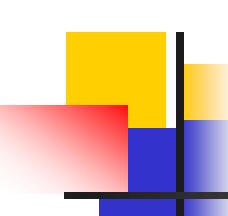
```
public Display getDisplay() {  
    return display;  
}  
public void pauseApp() {}  
public void destroyApp(boolean unconditional)  
{  
    System.gc();  
    notifyDestroyed();  
}  
private Image createImage(String filename) {  
    Image image = null;  
    try {  
        image = Image.createImage(filename);  
    } catch (Exception e) {  
    }  
    return image;  
}
```

```
public void mainMenuScreenShow(Alert alert)  
{  
    if (alert==null)  
        display.setCurrent(mainMenuScreen);  
    else  
        display.setCurrent(alert,mainMenuScreen);  
}  
public void mainMenuScreenQuit() {  
    destroyApp(true);  
}
```



Eliminator: Game Menu, EliminatorSubMenu (1)

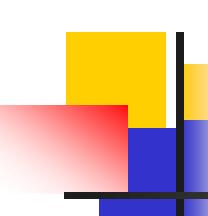
```
private void scnNewGame() {  
    midlet.mainMenuScreenShow();  
}  
  
private void scnSettings() {  
    midlet.settingsScreenShow();  
}  
  
private void scnHighScore() {  
    midlet.highScoreScreenShow();  
}  
  
private void scnHelp() {  
    midlet.helpScreenShow();  
}  
  
private void scnAbout() {  
    midlet.aboutScreenShow();  
}  
}
```



Eliminator: Game Menu, EliminatorSubMenu (2)

■ High Score Screen Source Code:

```
import javax.microedition.lcdui.*;  
public class HighScoreScreen extends Form implements CommandListener {  
    private Eliminator midlet;  
    private Command backCommand = new Command("Back", Command.BACK,1);  
    private Command resetCommand = new Command("Rest", Command.SCREEN,1);  
    public HighScoreScreen (Eliminator midlet) {  
        super("High Score");      this.midlet = midlet;  
        StringItem stringItem = new StringItem(null,"JL 100\nJL 50\nJL 10");  
        append(stringItem);      addCommand(backCommand);  
        addCommand(resetCommand);  setCommandListener(this); }  
    public void commandAction(Command c, Displayable d) {  
        if (c == backCommand) {  
            midlet.mainMenuScreenShow();  
            return;  
        }if (c == resetCommand) { // not implemented yet  
            System.out.println("Reset High Scores Not Implemented Yet");  
        }}}
```



Eliminator: Game Menu, EliminatorSubMenu (3)

■ Help Screen Source Code:

```
import javax.microedition.lcdui.*;
public class HelpScreen extends Form implements CommandListener {
private Eliminator midlet;
private Command backCommand = new Command("Back", Command.BACK, 1);
public HelpScreen (Eliminator midlet) {
super("Help");           this.midlet = midlet;
StringItem stringItem = new StringItem(null,"It is the year 3023, many things have changed over
the years " +
.....
);
append(stringItem);    addCommand(backCommand);
setCommandListener(this); }
public void commandAction(Command c, Displayable d) {
if (c == backCommand) {
midlet.mainMenuScreenShow();
return;
}}}
```



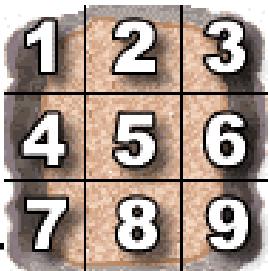
Eliminator: Game Menu, EliminatorSubMenu (4)

About Screen Source Code:

```
import javax.microedition.lcdui.*;  
  
public class AboutScreen extends Form implements CommandListener {  
    private Eliminator midlet;  
    private Command backCommand = new Command("Back", Command.BACK, 1);  
    public AboutScreen (Eliminator midlet) {  
        super("About");      this.midlet = midlet;  
        StringItem stringItem = new StringItem(null,"Eliminator\nVersion 1.0.0\nBy Jason Lam");  
        append(stringItem);  
        addCommand(backCommand);  setCommandListener(this);  
    }  
    public void commandAction(Command c, Displayable d) {  
        if (c == backCommand) {  
            midlet.mainMenuScreenShow();  
            return;  
        }}}
```

Eliminator: Terrain (Scrolling Background)

```
private TiledLayer loadTerrain() throws  
Exception {  
    Image tileImages =  
        Image.createImage("/terrain.png");  
    TiledLayer tiledLayer = new  
        TiledLayer(TILE_NUM_COL,TILE_NUM_  
        ROW,tileImages,TILE_WIDTH,TILE_HEI  
        GHT);  
    // Define Terrain Map  
    int[][] map = {  
        {0,0,0,0,0}, {3,0,0,0,0}, {6,0,0,0,0},  
        {6,0,0,0,1,2}, {6,0,0,0,4,5}, {6,0,0,0,7,8},  
        {6,0,0,0,0,0},{9,0,1,2,3,0}, {0,0,4,5,6,0},  
        {0,0,7,8,9,0},{0,0,0,0,0,0}, {0,0,0,0,0,0},  
        {0,0,0,0,0,0},{3,0,0,0,0,0}, {6,0,0,0,0,0},  
        {6,0,0,0,1,2},{6,0,0,0,4,5}, {6,0,0,0,7,8},  
        {6,0,0,0,0,0},{9,0,0,0,0,0}, {0,0,0,0,0,0},  
        {0,0,0,0,0,0},{0,0,0,0,0,0}, {3,0,0,0,0,1}};  
};
```



```
{6,0,0,0,1,2}, {6,0,0,0,4,5}, {6,0,0,0,7,8},  
{6,0,0,0,0,0}, {9,0,1,2,3,0}, {0,0,4,5,6,0},  
{0,0,7,8,9,0}, {0,0,0,0,0,0},{0,0,0,0,0,0},  
{0,0,0,0,0,0},{3,0,0,0,0,0}, {6,0,0,0,0,0},  
{6,0,0,0,1,2},{6,0,0,0,4,5}, {6,0,0,0,7,8},  
{6,0,0,0,0,0},{9,0,0,0,0,0}, {0,0,0,0,0,0},  
{0,0,0,0,0,0},{0,0,0,0,0,0}, {3,0,0,0,0,1}  
};  
// Map Terrain Map with actual graphic from terrain.png  
for (int row=0; row<TILE_NUM_ROW; row++) {  
    for (int col=0; col<TILE_NUM_COL; col++) {  
        tiledLayer.setCell(col,row,map[row][col]);  
    }  
}  
return tiledLayer;  
}  
■ Ví dụ: EliminatorScrolling
```

Eliminator: Player , ví dụ : EliminatorPlayer

■ Player Sprite

```
public class PlayerSprite extends Sprite {  
    private static final int MOVE = 3;  
    private int x,y;  
    private int scnWidth,scnHeight;  
    private int frameWidth, frameHeight;  
    private int frame;  
    private int lives;  
    public PlayerSprite(Image image, int frameWidth,  
        int frameHeight, int scnWidth, int scnHeight)  
        throws Exception {  
        super(image, frameWidth, frameHeight);  
        x = frameWidth/2;  
        y = frameHeight/2;  
        this.scnWidth = scnWidth;  
        this.scnHeight = scnHeight;  
        this.frameWidth = frameWidth;  
        this.frameHeight = frameHeight;  
        this.frame = 1;  this.lives = 3;}
```

```
    public void startPosition() {  
        setPosition(scnWidth/2,scnHeight/2);}  
    public void moveLeft() {  
        getXY();  
        if (x - MOVE > 0)  
            move(MOVE * -1,0);}  
    public void moveRight() {  
        getXY();  
        if (x + MOVE + frameWidth < scnWidth)  
            move(MOVE,0);}  
    public void moveUp() {  
        getXY();  
        if (y - MOVE > 0)  
            move(0,MOVE * -1);}  
    public void moveDown() {  
        getXY();  
        if (y + MOVE + frameHeight < scnHeight)  
            move(0,MOVE);}
```