Como desenvolver um jogo em Java ME - Parte 6

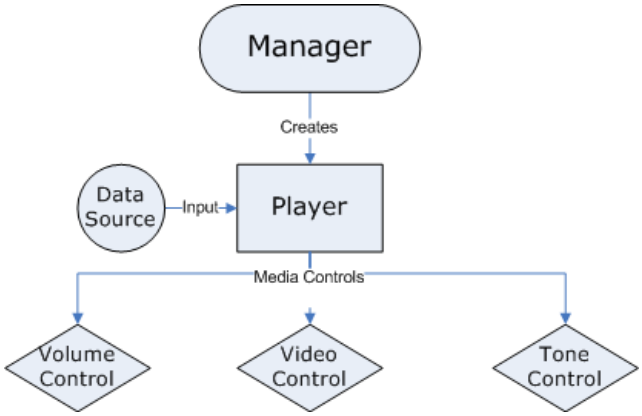
Em nossa última lição, aprendemos como salvar nossas configurações, criamos a tela Sound On/Off, etc. Mas por que temos esta opção, se não temos nenhum tipo de som em nosso jogo? É hora de aprender sobre a Mobile Media API (MMAPI) do Java ME e adicionar sons ao nosso jogo. Vamos lá!

A MMAPI oferece um conjunto de funcionalidades multimídia para dispositivos móveis, includindo playback e gravação de audio e video de uma variedade de fontes. Obviamente, nem todos os dispositivos oferecem as mesmas opções e suporte aos mesmos formatos, mas a MMAPI é desenhada de uma maneira que se aproveita de todas as vantagens das funcionalidades disponíveis, e ignorando aquelas que o aparelho não suporta.

MMAPI Info

A MMAPI é construída em um alto nível de abstração de todos os dispositivos multimídia. Esta abstração é implementada em três classes, que formam o núcleo de operações a serem feitas com esta API. Essas classes são: Manager, Player e a interface Control. Outra classe, DataSource, é usada para localizar recursos, mas a menos que você defina um novo jeito de ler dados você provavelmente não precisará utilizá-la diretamente.

Em suma, você usa a classe Manager para criar instâncias de Player para diferentes tipos de mídia especificando instâncias de DataSource. As instâncias de Player criadas são configuráveis através de instâncias de Control. Por exemplo, quase todos os Players em teoria suportam um VolumeControl para controlar o volume de áudio do Player. Veja o diagrama a seguir:



Então vamos começar com a classe Manager; basicamente é uma factory de Players que suporta os seguintes métodos de criação:

- createPlayer(DataSource source), cria um Player baseado em um DataSource
- createPlayer(InputStream stream, String type), cria um Player usando um InputStream como fonte e assumindo o típo de mídia provido . Para uma lista de típos de mídia veja o site IANA.
- createPlayer(String locator), cria um Player usando uma URL para identificar a fonte de dados.

O último médoto permite que você aloque diferentes tipos de mídia, depdendendo do protocolo URL que você escolha. Aqui está uma lista de tipos suportados:

- Midi Player - "device://midi", cria um Player MIDI.
- Tone Player - "device://tone", cria um Player de tones.
- Capture Audio - "capture://audio", permite captura de áudio do microfone.
- Capture Video - "capture://video", permite captura de vídeo da câmera.
- Capture Radio - "capture://radio?f=105.1&st=stereo", permite ouvir rádio.

Se você deseja saber quais tipos de conteúdo e protocolo são suportados pelo seu aparelho, use os seguintes métodos da classe Manager:

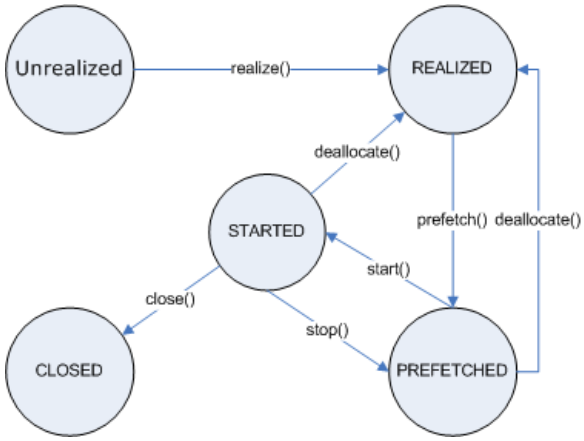
- getSupportedContentTypes(), fornece uma lista de tipos de conteúdo para todos os protocolos ou para um que você especificar.
- getSupportedProtocols(), fornece uma lista de protocolos disponívels para todos os tipos de conteúdo ou para um que você especificar.

Depois de criado um Player, você pode começar a usá-lo simplesmente chamando o método `start()`; quando ele chegar ao fim do fluxo de mídia, cessará automaticamente. Esta é uma visão mais simples da classe Player, mas na verdade ela possui 5 estados:

(C) Copyright Nokia 2014. All rights reserved.

- UNREALIZED, este é o estado inicial do Player obtido da classe Manager.
- REALIZED, quando o método realize() é chamado, o Player entra nesse estado obtendo as informações requeridas para adquirir os recursos de mídia. Realizar um Player pode ser um processo demorado e custoso. O Player pode ter que se comunicar com um servidor, ler um arquivo, ou interagir com um set de objetos.
- PREFETCHED, depois que um Player é realizado, ele ainda pode precisar adquirir recursos escassos ou exclusivos, preencher buffers com dados, ou executar alguma outra tarefa de processamento inicial. Isto é feito utilizando-se o método `prefetch()`, que troca o player para este estado.
- STARTED, quando `start()` é chamado, o Player começa a tocar o recurso de mídia até que este atinja o seu fim.
- CLOSED, quando `close()` é chamado o Player transiciona para este estado, liberando todos os recursos anteriormente adquiridos; neste ponto, o Player não pode ser reutilizado.

A imagem seguinte mostra todos os estados possíveis e as transições entre eles:



Se sua aplicação precisa de informações sobre as mudanças de estado do Player, você precisa implementar a interface `PlayerListener`.

Tocar um som

Agora que temos todas as informações básicas sobre a MMAPI, vamos começar a usá-la em nosso clone do Arkanoid. A idéia é tocar um som a cada vez que a bola atingir um tijolo ou a raquete. Para fazer isso, criaremos uma classe chamada Multimedia, com um método chamado playSound():

```
//multimedia libraries
import javax.microedition.media.Manager;
import javax.microedition.media.Player;
import javax.microedition.media.MediaException;
public class Multimedia {
  public void playSound(String file, String format) {
    try {
      InputStream is = getClass().getResourceAsStream(file);
      Player p = Manager.createPlayer(is, format);
      p.start();
    } catch (IOException ioe) {
    } catch (MediaException me) {
    }
  }
```

Agora precisamos apenas usar este método a cada vez que detectarmos uma colisão entre a bola e outras entidades. Eu criei, para este propósito, um arquivo de som chamado "click.wav", que adicionei à nossa pasta de recursos.

```
public void updateGameState(){
  ...
  byte colision = ball.colided(pad);
  if (colision != Entity.COLLISION_NONE){
      if (midlet.soundOn){
```


(C) Copyright Nokia 2014. All rights reserved.

```
        midlet.multimedia.playSound("click.wav", "audio/X-wav");
      }
    }
  ...
}
```

Se você rodar sua aplicação agora, você finalmente ouvir alguns sons em seu jogo.

Capturar vídeo

Agora que temos som, vamos adicionar outra feature interessante: vamos capturar uma foto do jogador cada vez que ele atinge uma pontuação alta. Para isso precisamos primeiro acessar a camera de vídeo e então mostrá-la ao jogador. O método a seguir captura uma foto da câmera para um Item:

```
  Player p;
  VideoControl vc;
  public Item showVideo(String url){
    Item result = null;
    try {
        p = Manager.createPlayer(url);
        p.realize();
        // Grab the video control .
        vc = (VideoControl)p.getControl("VideoControl");
        if (vc != null) {
            // create the Item with the video image
            result =((Item)vc.initDisplayMode(VideoControl.USE_GUI_PRIMITIVE, null));
            // add a label
            result.setLabel("Photo");
        }
        // start capture
        p.start();
    } catch (IOException ioe) {
    } catch (MediaException me) { }
    return result;
  }
```

Como você pode ver, estamos usando um Control (VideoControl) para criar o Item a ser usado em nosso Form.

```
public Displayable initNewHighScore(int score, int pos) {
  ...
  newHighScoreForm.append(multimedia.showVideo("capture://video"));
  ...
}
```

Agora usamos o VideoControl para capturar uma foto da câmera:

```
  public Image captureVideo(){
    Image result = null;
    try {
        // grab data
        byte[] imageData = vc.getSnapshot("encoding=png");
        // create image;
        result = Image.createImage(imageData, 0, imageData.length);
    } catch (MediaException me) {
      me.printStackTrace();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }
    return result;
  }
```

e então chamamos o método abaixo para salvar a pontuação:

```
  // we added an extra field to Score to store the image
  scores[pos].image = multimedia.captureVideo();
```

Depois, precisamos apenas mostrar nossas imagens na tela de pontuações:

```
public Displayable initScoreForm() {
...
  if (scores[i].image != null){
    highScoreForm.append(scores[i].image);
  }
...
}
```

Agora rode a aplicação para testar a nova feature! Com isso encerramos a nossa lição; na próxima aprenderemos a usar a funcionalidade de rede de seu aparelho.

Downloads:

- Full Source Code
- Jad File
- Jar File

(C) Copyright Nokia 2014. All rights reserved.