Como desenvolver um jogo em Java ME - Parte 5

Após nossa última lição, temos um midlet Arkanoid completamente funcional, mas ainda falta uma coisa muito importante para um jogo: salvar as maiores pontuações! Até agora todas elas eram perdidas assim que saíamos da aplicação (o mesmo acontece com as configurações do jogo). Por que jogar um jogo se você não pode bater recordes e mostrar a seus amigos?!!!

Para implementar esta funcionalidade, você precisa entender como Input/Output funcionam em um midlet e então como usar RecordStores.

Streams

Vamos começar com simples operações de IO. As classes que as implementam estão localizadas no pacote java.io. Java ME tem apenas uma fração das classes disponíveis em Java SE mas elas são bastante úteis:

- **Input Stream, Output Stream**: as classes básicas para fluxos binários
- **ByteAraryInputStream, ByteArrayOutputStream**: fluxos para bufferizar arrays na memória
- **DataInputStream, DataOutputStream**: ler e escrever tipos Java primitivos (int, float, String, ...) em streams
- **Reader, Writer**: as classes básicas de fluxos de caracteres
- **OutputStreamWriter, InputStreamReader**: classes para leitura de caracteres usando encodings

Para exemplificar o uso dessas classes vamos criar um arquivo de configurações onde guardaremos algumas das opções do jogo:

- Vidas
- Velocidade da bola
- Tempo para completar uma fase
- Número de pontos ganhos para cada tijolo destruído

Para implementar esse arquivo, criaremos um simples arquivo de texto, chamado "settings.txt", onde cada linha representa uma configuração, com o nome e seu valor separados por um caractere "=".

```
ball_speed=2
start_lifes=4
level_time=100
brick_points=10
```

Depois de criar este arquivo, adicione-o aos recursos do seu projeto e certifique-se de que o arquivo será compactado dentro do arquivo .jar juntamente com suas classes.

Para acessar o arquivo, você precisa usar o método `getResourceAsStream(String name)` da classe Class. Este método lhe dá acesso a qualquer arquivo armazenado dentro do arquivo JAR. Por exemplo, para acessar o arquivo "settings.txt", você pode usar este código:

```
InputStream is = this.getClass().getResourceAsStream("/settings.txt");
// read first byte
byte c= is.read();
```

Este código lhe dá o primeiro byte do arquivo, mas o que você precisa fazer é ler cada linha como um fluxo de caracteres, e para isso usaremos a classe InputStreamReader e então fazer o parsing de cada linha procurando pelo nome e valor das configurações.

```
public class Settings {
  public Hashtable values;

  public Settings(String file) {
    values = new Hashtable(10);
    read(file);
  }

  public String getSetting(String key){
    return (String)(values.get(key));
  }

  /**
   * Opens the file and reaas all the settings to the hashtable
   * @param file, the name of the file to read
   */
  public void read(String file){
    // open file
    InputStream is = this.getClass().getResourceAsStream(file);
    InputStreamReader isr = new InputStreamReader(is);
    // create a buffer to store lines
    StringBuffer lineBuffer = new StringBuffer();
    int c;
    try {
      c = isr.read();
      while(c != -1){
        lineBuffer.append((char)c);
        c = isr.read();
        // checks for end of line character
        if ( c == 10 || c==-1){
          // cleans extra spaces or end of lines chars
          String line = lineBuffer.toString().trim();
          // splits the string using the = character
          int pos = line.indexOf("=");
          if (pos != -1){
            // adds a new setting
            String key = line.substring(0,pos);
            String value = line.substring(pos+1);
            values.put(key, value);
          }
          // clean buffer
```


(C) Copyright Nokia 2013. All rights reserved.

```
          lineBuffer.setLength(0);
        }
      }
    } catch (IOException e) {
      e.printStackTrace();
    } finally {
      try {
        is.close();
      } catch (IOException e) {
        e.printStackTrace();
      }
    }
  }
}
```

Agora precisamos apenas usar esta classe em nosso Canvas, dentro do método init(), para acessar as configurações do jogo:

```
public void initSettings(){
   Settings setting = new Settings("/settings.txt");
   BALL_SPEED = Integer.parseInt(setting.getSetting("ball_speed"));
   MAX_LIFES = Integer.parseInt(setting.getSetting("start_lifes"));
   MAX_TIME = Integer.parseInt(setting.getSetting("level_time"));
   BRICK_SCORE = Integer.parseInt(setting.getSetting("brick_points"));
 }

public void init(){
  initSettings();
  [...]
}
```

Ok, agora temos um arquivo de configuração, mas ainda não descobrimos como salvar as pontuações. O método getResourceAsStream() permite apenas a leitura de arquivos armazenados no arquivo .jar, não a escrita nos mesmos.

RecordStore

Para escrever em arquivos você precisa usar a classe RecordStore, pois ela permite que você crie mini-bancos de dados onde você pode ler e escrever dados, que são persistentes entre cada execução do midlet.=

A classe é implementada no pacote javax.microedition.rms e fornece os seguintes construtores estáticos:

- openRecordStore(String recordStoreName, boolean createIfNecessary)
- openRecordStore(String recordStoreName, boolean createIfNecessary, int authmode, boolean writable)
- openRecordStore(String recordStoreName, String vendorName, String suiteName)

Todos esses métodos permitem a criação ou abertura de um record store. O nome de um record store é case-sensitive e pode ter um máximo de 32 caracteres. Uma característica especial é o uso do *authmode*. Esta configuração possui dois valores:

- **AUTHMODE_PRIVATE**, neste modo apenas o midlet que criou o record store pode ler e escrever nele
- **AUTHMODE_ANY**, neste modo qualquer midlet pode abrir o record store, ler seus dados e se o flag *writable* estiver definido, escrever dados de volta no record store. Isto permite o compartilhamento de dados entre midlets; por exemplo, você pode ter um cenário onde dois midlets compartilham a tabela de pontuação. Os record stores são unicamente identificados pelo seu nome e pelas propriedades Midlet-Name,Midlet-Vendor definidas no arquivo Jad.

Após abrir um RecordStore, você pode adicionar records a ele. Cada record é composto de um vetor de bytes e é identificado por um id único criado no momento da primeira escrita. Após criar um record você pode ler seus dados, modificá-los e deletar o record. Os métodos seguintes suportam essas operações:

- addRecord(byte[] data, int offset, int numBytes)
- deleteRecord(int recordId)
- setRecord(int recordId, byte[] newData, int offset, int numBytes)

O tamanho máximo de um RecordStore depende do dispositivo; você pode usar o método getSizeAvailable() para ter um valor-base, mas muitos dispositivos retornam valores não confiáveis.

Agora que entendemos a classe RecordStore melhor, vamos usá-la para salvar nossas pontuações:

```
public void saveData() {
  try {
    // open records store options
    RecordStore options = RecordStore.openRecordStore("options", true);
    byte[] data = saveOptions();
    // check if record store not empty
    if (options.getNumRecords() != 0) {
      // update the settings
      options.setRecord(1, data, 0, data.length);
    } else {
      // adds the settings
      options.addRecord(data, 0, data.length);
    }
    // closes the record store
    options.closeRecordStore();
  } catch (RecordStoreException ex) {
  }
}

public byte[] saveOptions() {
  // create a byte array stream to store data temporarily
  ByteArrayOutputStream baos = new ByteArrayOutputStream();
  DataOutputStream dos = new DataOutputStream(baos);
  try {
    dos.writeBoolean(soundOn);
    // write scores
    for (int i =0; i < scores.length; i++){
      dos.writeInt(scores[i].value);
      dos.writeUTF(scores[i].name);
      dos.writeLong(scores[i].when.getTime());
    }
    // push all the data to the byte array stream
    dos.flush();
  } catch (IOException ex) {
```


(C) Copyright Nokia 2013. All rights reserved.

```
    }
    // returns bytes from stream
    return baos.toByteArray();
  }
```

Como você pode observar, estamos criando um Record Store de nome "options" e usando-o para armazenar nossas configurações e pontuações. Para o processo de gravação estamos usando um DataOutputStream com um ByteArrayOutputStream para facilitar nosso trabalho de converter todos os dados para um vetor de bytes.

Agora implementamos a leitura de dados:

```
  public void loadData() {
    try {
      RecordStore options = RecordStore.openRecordStore("options", true);
      // check if record store not empty
      if (options.getNumRecords() != 0) {
        loadOptions(options.getRecord(1));
      }
      options.closeRecordStore();
    } catch (RecordStoreException ex) {
    }
  }

  public void loadOptions(byte[] data) {
    // create a byte array stream to store data temporarily
    ByteArrayInputStream bais = new ByteArrayInputStream(data);
    // creates a data input stream to read from
    DataInputStream dis = new DataInputStream(bais);
    try {
      soundOn = dis.readBoolean();
      // read scores
      for (int i = 0; i < scores.length; i++) {
        int value = dis.readInt();
        String name = dis.readUTF();
        Date date = new Date(dis.readLong());
        scores[i] = new Score(value, name, date);
      }
      dis.close();
    } catch (IOException ex) {

    }
  }
```

Simplesmente abrimos o record store e checamos se ele possui algum record disponível para leitura. Se este for o caso, lemos os dados com um DataInputStream combinado com um ByteArrayInputStream.

Agora apenas adicionaremos chamadas para estes métodos em nosso midlet, tanto no início quando no final do mesmo:

```
  public void initOptions() {
    soundOn = true;
    if (scores == null) {
      scores = new Score[10];
      for (int i = 0; i < scores.length; i++) {
        scores[i] = new Score(0, "Empty", new Date());
      }
    }
    // loads data in record stores if available
    loadData();
  }

  public void exit() {
    // store high scores and setting to record store
    saveData();
    notifyDestroyed();
  }
```

Após estas últimas mudanças, finalmente temos nosso jogo salvando as melhores pontuações e as configurações. Na próxima lição finalmente colocamos em uso as configurações de jogo, e também adicionaremos som ao mesmo.

Downloads:

- Full Source Code
- Jad File
- Jar File

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.