Como desenvolver um jogo em Java ME - Parte 2

Já que a interação com o usuário é uma das preocupações principais em qualquer aplicação móvel, devido ao tamanho das telas, é importante que você entenda o básico desta faceta dos MIDlets. Neste artigo, você aprenderá quais elementos de UI estão disponíveis em um MIDlet e como usá-los para criar a interface para o nosso clone do jogo Arkanoid.

Qualquer interação com um usuário é feita através de um componente de UI. De fato, quando você criou o simples "Hello World" na parte 1 deste artigo, você usou um elemento chamado Alert para mostrar uma mensagem na tela. Esta mensagem foi mostrada na tela na realidade com a ajuda de outro elemento de UI chamado Display.

Vamos iniciar com uma discussão da arquitetura completa dos componentes gráficos.

Arquitetura

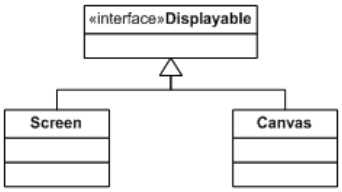
O MIDP 2.0 fornece classes gráficas em um pacote, javax.microedition.lcdui. Se você está se perguntando o que "LCDUI" significa, a resposta é "Liquid Crystal Display User Interface (LCD UI)". Para que você possa mostrar um elemento gráfico na tela, você precisa usar uma classe que implemente a interface `Displayable`. Tal classe pode ter um título, um ticker e certos comandos associados a si, entre outras características.

A classe Display gerencia o que está sendo mostrado na tela. O método estático getDisplay(MIDlet midlet) lhe fornece acesso ao Display do seu midlet. Assim você pode usar o método setCurrent(Displayable d) para escolher o que mostrar na tela em dado momento. VOcê pode ter apenas um Displayable sendo mostrado de cada vez. Lembra-se do código de nossa última lição?

```
Display.getDisplay(this).setCurrent(alert);
```

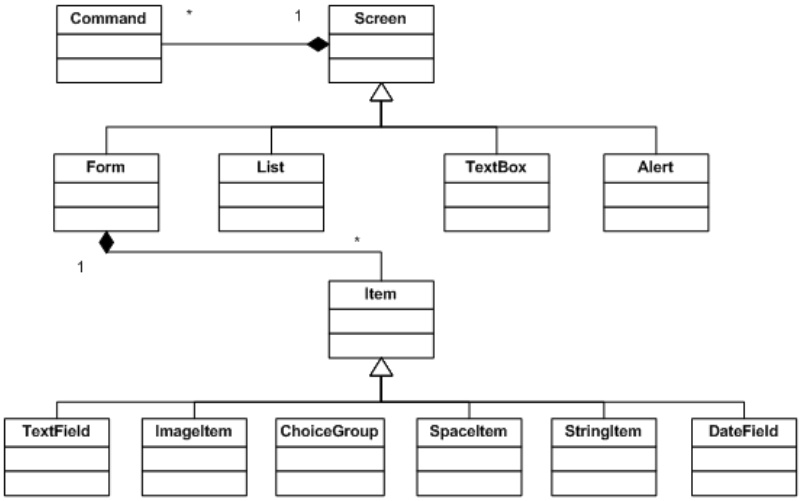
As classes Displayable do MIDP 2.0 pacote javax.microedition.lcdui podem ser divididas em dois grupos: as de alto nível e as de baixo nível.

O grupo de alto nível é implementado através da classe Screen, e o de baixo nível através da classe Canvas; ambas essas classes e suas subclasses implementam a interface Displayable, como pode ser visto na figura abaixo:



Interface de alto nível

As classes do grupo de alto nível são perfeitas para o desenvolvimento de MIDlets que devem rodar no maior número de devices possível, pois as mesmas não fornecem controle exato sobre o modo com o qual são mostrada. Ao contrário, elas são altamente abstratas, para permitir que o dispositivo em si gerencie a melhor forma de mostrá-las na tela, de acordo com suas capacidades de hardware. Abaixo você pode ver um diagrama de classes das mesmas:



Como você pode ver, existem várias classes que fornecem elementos gráficos, então vamos analisar cada uma delas.

Command

Um MIDlet interage com o usuário através de comandos. Um comando é o equivalente a um item de menu em uma aplicação desktop, e pode somente ser associado a um único elemento de UI. A classe Displayable permite que o usuário adicione um comando a ela utilizando o método addCommand(Command command). Um elemento Displayable pode ter múltiplos comandos associados a si.

(C) Copyright Nokia 2013. All rights reserved.

A classe Command contém informações sobre um comando. Esta informação é encapsulada em quatro propriedades, a saber: um label curto, um longo, um tipo de comando, e uma prioridade. Em nossa aplicação "HelloWorld" criamos um comando fornecendo esses valores no seu construtor:

```
// adds a command to exit the Midlet
comExit = new Command("Exit", Command.EXIT, 1);
```

Note que comandos são imutáveis, uma vez criados.

Especificar um tipo de comando permite que o device rodando o MIDlet mapeie quaisquer teclas predefinidas no device àquele comando. Por exemplo, um comando cujo tipo seja "OK" será mapeado à tecla "OK"do aparelho. Os outros tipos são: BACK, CANCEL, EXIT, HELP, ITEM, SCREEN e STOP. O tipo SCREEN relaciona-se com um comando definido para aplicação para a tela atual. Tanto SCREEN quanto ITEM provavelmente nunca serão mapeados a uma tecla específica do aparelho, mas sim apenas quando estiverem em destaque.

Para receber feedback do acionamento dos comandos, seu midlet precisa implementar a interface CommandListener, e o método commandAction(). Já fizemos isso em nosso midlet HelloWorld:

```
alert.addCommand(comExit);
// adds a listener to the form
alert.setCommandListener(this);
[...]
public void commandAction(Command cmd, Displayable display) {
  if (cmd == comExit) {
    exit();
  }
}
```

Como você pode ver, o método commandAction() recebe dois parâmetros: o Comando sendo executado e o Displayable atualmente sendo mostrado na tela.

Alert

Como você se lembra, utilizamos um alerta em nosso midlet Hello World; este elemento representa uma tela que mostra alguma mensagem ao usuário e aguarda um período de tempo antes de proceder para o próximo Displayable. Um alerta pode conter uma string de texto, um título e uma imagem. O uso ideal de um Alert é para informar ao usuário sobre erros ou alguma condição excepcional.

TextBox

A classe TextBox é uma Screen que permite ao usuário entrar e editar texto. Este elemento pode ser adaptado para suas necessidades especiais. Você pode restringir o número máximo de caracteres que o usuário pode digitar , e também o tipo de texto que é aceito. Existem seis flags para restringir o conteúdo: ANY, EMAILADDR, NUMERIC, PHONENUMBER, URL, and DECIMAL. ANY permite todos os tipos de texto, enquanto as outras flags atuam de acordo com seus nomes.

De maneira similar, existem seis flags que afetam o display em si. Estas são: PASSWORD, UNEDITABLE, SENSITIVE, NON_PREDICTIVE, INITIAL_CAPS_WORD, and INITIAL_CAPS_SENTENCE. Por exemplo, para aceitar apenas endereços de e-mail em um textbox, você precisa definir a flag TextField.EMAILADDR usando o método setConstraints(). Para fazer este campo de texto ser apenas de leitura, você precisa combinar a flag com TextField.UNEDITABLE. Isto é feito utilizando-se uma operação bitwise OR entre as duas, como se segue: setConstraints(TextField.EMAILADDR | TextField.UNEDITABLE);.

Para definir o conteúdo de um textbox, você pode usar o médoto setString() para definir o conteúdo completo, ou insert() para inserir mais texto em adição ao que já está digitado. A listagem mais abaixo (a terceira) mostra como usar ambos os métodos:

List

Uma lista contém um conjunto de escolhas. Quando uma lista está presente no Display, o usuário pode interagir com a mesma selecionando elementos e rolando através deles. A lista pode ser configurada para ser:

- Choice.EXCLUSIVE - apenas um elemento pode ser selecionado
- Choice.MULTIPLE - múltiplos elementos podem ser selecionados
- Choice.IMPLICIT - o elemento em destaque é automaticamente selecionado

Form

Um Form é uma Screen que contém uma mistura arbitrária de itens. Em geral, qualquer subclasse da classe Item pode estar contida dentro de um Form. A implementação ajusta o layout, traversal e scrolling. Todo o conteúdo de um form é rolado (scrolled) de forma conjunta.

Existem oito tipos de Item que podem ser adicionados a um Form:

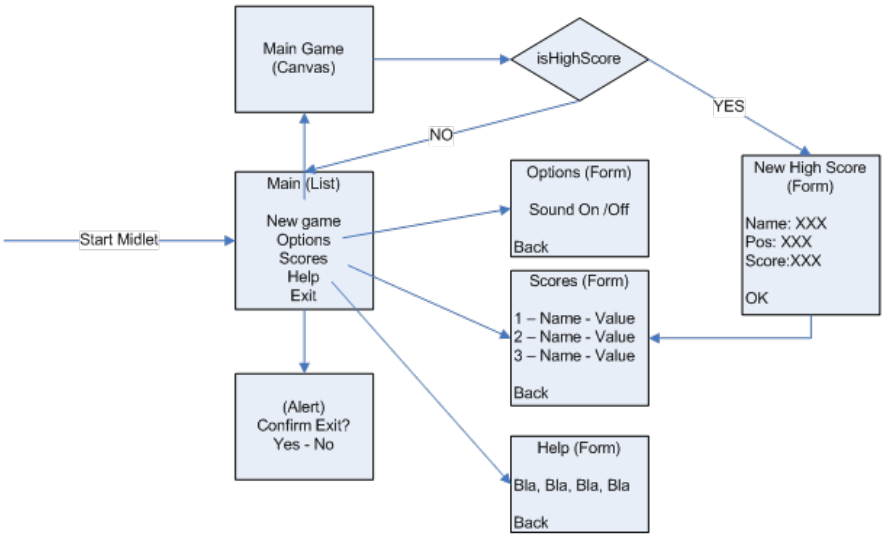
- **StringItem**, um rótulo que não pode ser modificado pelo usuário. Este item pode conter um título e texto, e ambos podem ser nulos para marcar o local como um placeholder. A classe Form provê um atalho para se adicionar um StringItem sem título: append(String text)
- **DateField**, permite que o usuário entre valores de dados e tempo em um dos três formatos: DATE, TIME, or DATE_TIME.
- **TextField**, o mesmo que um TextBox.
- **ChoiceGroup** o mesmo que uma List.
- **Spacer**, usado para posicionar elementos de UI através da colocação de alguns espaços entre eles. Este elemento é invisível e pode ter seu tamanho definido para qualquer valor.
- **Gauge**, é usado para simular uma barra de progresso. Entretanto, esta barra de progresso também pode ser usada de modo interativo pelo usuário, por exemplo como um controle de volume.
- **ImageItem**, um item que possui uma imagem. Como o StringItem, a classe Form também fornece um atalho para adicionar uma imagem:: append(Image image). Mais sobre imagem em uma seção posterior.

 (C) Copyright Nokia 2013. All rights reserved.

- **CustomItem**, é uma classe abstrata que permite a criação de subclasses de Item que tenha sua própria aparência customizável, assim como uma interatividade e mecanismos de notificação. Se você deseja um componente gráfico que seja muito diferente dos elementos fornecidos por padrão, você pode criar um CustomItem e adicioná-lo a um Form.

Exemplos de interface gráfica

Para aprender a usar todas essas classes, vamos criar uma interface simples para o nosso jogo Arkanoid. Implementaremos as seguintes telas:



Para cada uma das telas criaremos um método init[ScreenName] que inicializará cada uma das telas e retornará o elemento Displayable criado.=

Usaremos o componente List para mostrar as opções principais; veja o código abaixo:

```
public Displayable initMainForm() {
  if (mainForm == null) {
    // creates a implicit List where the   current element is
    // the selected
    mainForm = new List("Menu",   List.IMPLICIT);
    // append list options
    mainForm.append("New Game",   null);
    mainForm.append("Options",   null);
    mainForm.append("Scores",   null);
    mainForm.append("Help", null);
    mainForm.append("Exit", null);

    // adds a select Command
    comSelect = new   Command("Select", Command.ITEM, 1);
    mainForm.setSelectCommand(comSelect);

    // adds a listener to the form
    mainForm.setCommandListener(this);
  }
    return mainForm;
}
```

Para o menu de configurações escolhemos um elemento Form onde adicionamos um ChoiceGroup para opções de Som (On-Off)

```
public Displayable initSettingsForm() {
  // check if already created
  if (settingsForm == null) {
    settingsForm = new Form("Settings");
    settingsForm.addCommand(initBackCommand());
    settingsForm.setCommandListener(this);
    // creates a choice Group for sound options
    soundChoice = new ChoiceGroup("Sound", List.EXCLUSIVE);
    soundChoice.append("On", null);
    soundChoice.append("Off", null);
    // appends the choice to the form
    settingsForm.append(soundChoice);
  }

  return settingsForm;
}
```

Para a tela de ajuda escolhemos um simples Form com uma mensagem estática:

```
public Displayable initHelpForm() {
    if (helpForm == null) {
      helpForm = new Form("Help");
      helpForm
          .append("User cursors to move your pad, don't let "+
              "the ball go by you, hit all the bricks!");
      helpForm.setCommandListener(this);
      helpForm.addCommand(initBackCommand());
    }
    return helpForm;
  }
```

Para introduzir um novo high score usaremos um Form com um TextField e um Date Field:

```
public Displayable initNewHighScore(int score, int pos) {
    if (newHighScoreForm == null) {
```


(C) Copyright Nokia 2013. All rights reserved.

```
      newHighScoreForm = new Form("New High Score");
      newHighScoreForm.setCommandListener(this);
      // create items
      highScoreName = new TextField("Name", "", 20, TextField.ANY);
      highScoreValue = new StringItem("Score", Integer.toString(score));
      highScorePosition = new StringItem("Position", Integer.toString(pos));
      // create save command
      highScoreSave = new Command("Save", Command.OK, 1);
      // append command and itens to screen
      newHighScoreForm.addCommand(highScoreSave);
      newHighScoreForm.append(highScoreName);
      newHighScoreForm.append(highScoreValue);
      newHighScoreForm.append(highScorePosition);
    }
    // update score
    highScoreValue.setText(Integer.toString(score));
    // update pos
    highScorePosition.setText(Integer.toString(pos)+1);
    return newHighScoreForm;
  }
```

Deixaremos a tela de jogo para a próxima lição, mas no momento vamos criar um método que simula o final do jogo e usá-lo:

```
  public void endGame(int lifes, int score, int time) {
    Displayable nextScreen = initMainForm();
    String message;
    if (lifes == 0) {
      message = "Game Over!!";
    } else {
      message = "You Win!";
    }
    int pos = isHighScore(score);
    if (pos != -1) {
      nextScreen = initNewHighScore(score, pos);
    }

    display(new Alert(message, message, null, AlertType.INFO), nextScreen);
  }
```

Agora que temos todas as nossas telas, precisamos ligá-las ao método commandAction(). Vamos reescrever nosso código:

```
public void commandAction(Command cmd, Displayable display) {
  // check what screen is being displayed
  if (display == mainForm) {
    // check what command was used
    if (cmd == comSelect) {
      switch (mainForm.getSelectedIndex()) {
      case (0):
        // At the moment just go directly to the end of the game
        endGame(1, 200, 50);
        break;
      case (1):
        display(initSettingsForm());
        break;
      case (2):
        display(initScoreForm());
        break;
      case (3):
        display(initHelpForm());
        break;
      case (4):
        exit();
        break;
      }
    }
  } else if (display == highScoreForm) {
    if (cmd == comBack) {
      display(initMainForm());
    }
  } else if (display == settingsForm) {
    if (cmd == comBack) {
      soundOn = soundChoice.getSelectedIndex() == 0;
      display(initMainForm());
    }
  } else if (display == helpForm) {
    if (cmd == comBack) {
      display(initMainForm());
    }
  } else if (display == newHighScoreForm) {
      if (cmd == highScoreSave) {
        int pos = Integer.parseInt(highScorePosition.getText())-1;
        // advance all the scores
        for ( int i = scores.length-1; i > pos ; i--){
          scores[i].name  = scores[i-1].name;
          scores[i].value = scores[i-1].value;
          scores[i].when  = scores[i-1].when;
        }
        // insert new score
        scores[pos].name = highScoreName.getString();
        scores[pos].value = Integer.parseInt(highScoreValue.getText());
        scores[pos].when = new Date();
        display(initScoreForm());
      }
    }
}
```

Dentro do método commmandAction() você especifica toda a lógica do menu para nosso midlet, decidindo o que fazer dependendo de qual displayable está sendo mostrado e de qual comando foi selecionado. A partir do menu principal simplesmente redirecionamos o usuário para cada tela específica. Para as outras telas, temos apenas um comando, "Back", por enquanto. A única exceção é a tela NewHighScores onde um comando "Save" guarda os high scores em um vetor.

Você pode ter notado o uso do método display(), que é apenas um simples helper para ativar um Displayable.

```
public void display(Displayable display) {
  // shows display  in the screen.
  Display.getDisplay(this).setCurrent(display);
}
```


(C) Copyright Nokia 2013. All rights reserved.

Agora rode seu midlet e finalmente você terá sua primeira interface gráfica. Na próxima lição, implementaremos a tela Game Screen.

Downloads:

- Full Source Code
- Jad File
- Jar File

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.