# Módulo 5

Desenvolvimento de Aplicações Móveis


# Lição 4

Interface de Baixo Nível para o Usuário

*Versão 1.0 - Set/2007*

**Autor**
xxx

**Equipe**
Rommel Feria
John Paul Petines

## Necessidades para os Exercícios

### Sistemas Operacionais Suportados

**NetBeans IDE 5.5** para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

**NetBeans Enterprise Pack**, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

### Configuração Mínima de Hardware

**Nota:** IDE NetBeans com resolução de tela em 1024x768 pixel

| Sistema Operacional | Processador | Memória | HD Livre |
|---|---|---|---|
| Microsoft Windows | 500 MHz Intel Pentium III workstation ou equivalente | 512 MB | 850 MB |
| Linux | 500 MHz Intel Pentium III workstation ou equivalente | 512 MB | 450 MB |
| Solaris OS (SPARC) | UltraSPARC II 450 MHz | 512 MB | 450 MB |
| Solaris OS (x86/x64 Platform Edition) | AMD Opteron 100 Série 1.8 GHz | 512 MB | 450 MB |
| Mac OS X | PowerPC G4 | 512 MB | 450 MB |

### Configuração Recomendada de Hardware

| Sistema Operacional | Processador | Memória | HD Livre |
|---|---|---|---|
| Microsoft Windows | 1.4 GHz Intel Pentium III workstation ou equivalente | 1 GB | 1 GB |
| Linux | 1.4 GHz Intel Pentium III workstation ou equivalente | 1 GB | 850 MB |
| Solaris OS (SPARC) | UltraSPARC IIIi 1 GHz | 1 GB | 850 MB |
| Solaris OS (x86/x64 Platform Edition) | AMD Opteron 100 Series 1.8 GHz | 1 GB | 850 MB |
| Mac OS X | PowerPC G5 | 1 GB | 850 MB |

### Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

### *Colaboradores que auxiliaram no processo de tradução e revisão*

Aécio Júnior
Alexandre Mori
Alexis da Rocha Silva
Allan Souza Nunes
Allan Wojcik da Silva
Anderson Moreira Paiva
Andre Neves de Amorim
Angelo de Oliveira
Antonio Jose R. Alves Ramos
Aurélio Soares Neto
Bruno da Silva Bonfim
Carlos Fernando Gonçalves
Denis Mitsuo Nakasaki
Fábio Bombonato
Fabrício Ribeiro Brigagão
Francisco das Chagas
Frederico Dubiel
Herivelto Gabriel dos Santos
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Kefreen Ryenz Batista Lacerda
Kleberth Bezerra G. dos Santos
Leandro Silva de Morais
Leonardo Ribas Segala
Lucas Vinícius Bibiano Thomé
Luciana Rocha de Oliveira
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Mauro Cardoso Mortoni
Paulo Afonso Corrêa
Paulo Oliveira Sampaio Reis
Pedro Henrique Pereira de Andrade
Ronie Dotzlaw
Seire Pareja
Sergio Terzella
Vanessa dos Santos Almeida
Robson Alves Macêdo

## *Auxiliadores especiais*

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

## *Coordenação do DFJUG*

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

## Agradecimento Especial

**John Paul Petines –** Criador da Iniciativa JEDI™
**Rommel Feria** – Criador da Iniciativa JEDI™

# 1. Objetivos

Nas lições anteriores dissertamos sobre como construir interfaces do usuário tais como listas, formulários e dados de entrada. Aquelas eram interfaces de alto nível e o programador não tinha que se preocupar sobre como desenhar a tela ou posicionar o texto na tela. Todo programa tem que especificar apenas o tipo de componentes e o nomes dos elementos. O sistema encarrega-se de manusear o desenho, a rolagem e o *layout* da tela.

Uma desvantagem de se utilizar somente componentes de alto nível é que o programa não tem controle total sobre a tela. Há ocasiões em que queremos desenhar linhas, animar imagens e ter controle sobre cada ponto da tela.

Nesta lição trabalharemos diretamente com o desenho da tela. Estudaremos a classe *Canvas*, que deve ser o fundo de nosso desenho. Trabalharemos também com a classe *Graphics* que tem métodos para desenho de linhas, retângulos, arcos e texto. Também falaremos sobre fontes, cores e imagens.

Ao final desta lição, o estudante será capaz de:

- Compreender os eventos de baixo nível manuseados em MIDP
- Desenhar e exibir texto, imagens, linhas, retângulos e arcos
- Especificar cor, fonte e movimento para operações de desenho
- Compreender e utilizar as classes *Canvas* e *Graphics*

# 2. Canvas

A classe *Canvas* é uma subclasse de *Displayable*. Uma classe abstrata que deve ser instanciada antes que uma aplicação possa fazer uso de suas funcionalidades.

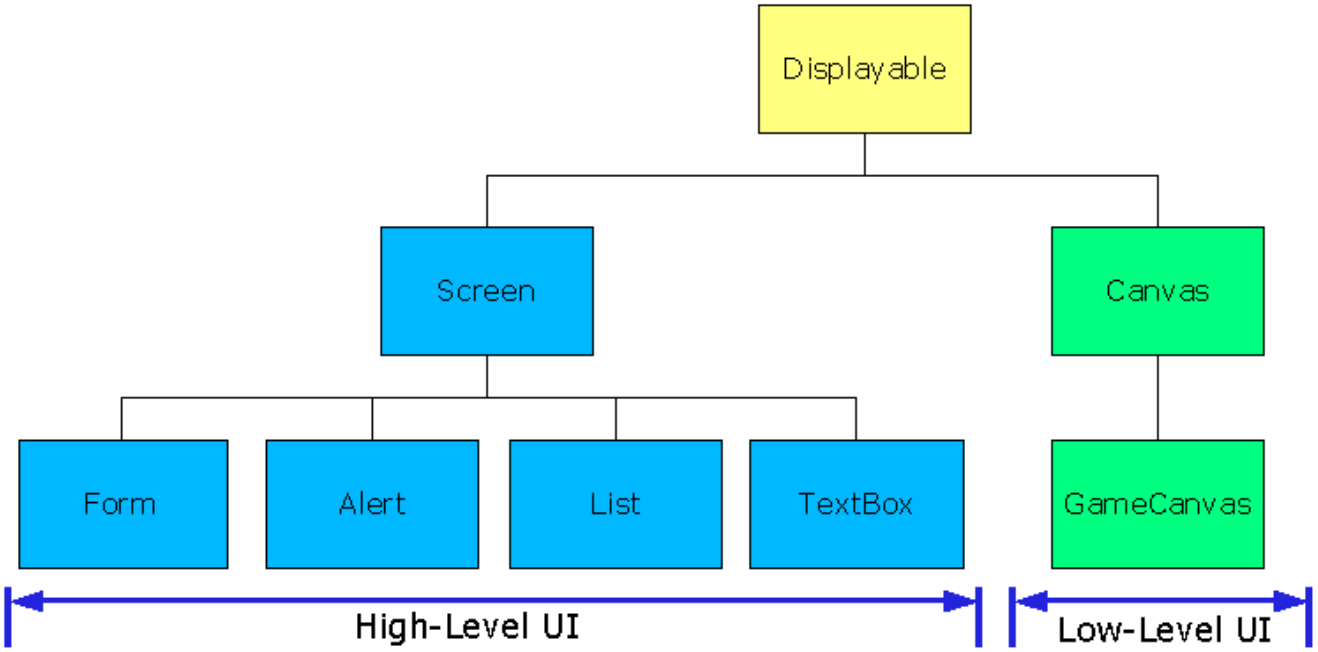


*Figura 1:* Hierarquia de classes.

Pode ser implementada com a subclasse *Screen* de exibição. O programa pode alternar para *Canvas* e *Screen*. Define métodos vazios de manuseio. As aplicações devem implementá-los para manusear os eventos.

A classe define um método abstrato chamado *paint*(). Aplicações que utilizam esta classe devem providenciar a implementação deste método.

## 2.1. O sistema de Coordenadas

O sistema de coordenadas da classe *Canvas* é baseado no ponto de coordenada zero, ou seja, as coordenadas x e y iniciam com o valor zero. O canto esquerdo superior é a coordenada (0,0). A coordenada x é incrementada da esquerda para a direita, enquanto a coordenada y é incrementada de cima para baixo. Os métodos *getWidth*() e *getHeight*() retornam a largura e a altura do *Canvas*, respectivamente.

O canto inferior direito da tela tem as coordenadas nas posições *getWidht*()-1 e *getHeight*()-1. Quaisquer alterações no tamanho de desenho do *Canvas* é registrado para a aplicação pelo método *sizeChange*(). O tamanho disponível do *Canvas* pode mudar se acontecer uma troca entre os modos normal e tela cheia ou a adição ou remoção de componentes tais como Comandos.

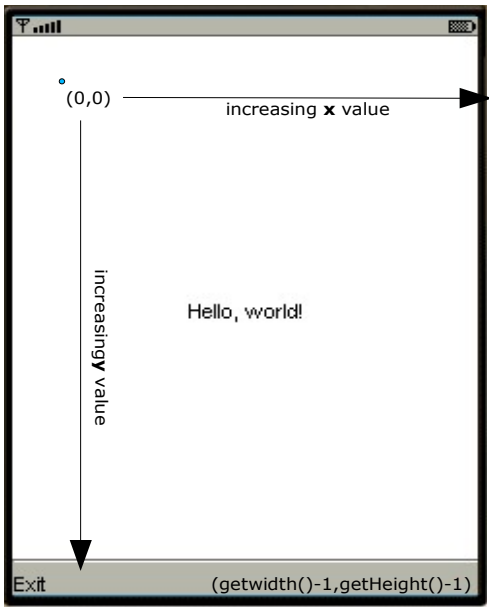


*Figura 2: O sistema Coordinate*

## 2.2. "Hello, world!"

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

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

public class HelloCanvasMIDlet extends MIDlet {
    private Display display;
    HelloCanvas canvas;
    Command exitCommand = new Command("Exit", Command.EXIT, 0);

    public void startApp() {
        if (display == null){
            canvas = new HelloCanvas(this, "Hello, world!");
            display = Display.getDisplay(this);
        }
        display.setCurrent(canvas);
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    protected void Quit(){
        destroyApp(true);
        notifyDestroyed();
    }
}

class HelloCanvas extends Canvas implements CommandListener {
    private Command exitCommand = new Command("Exit", Command.EXIT, 0);
    private HelloCanvasMIDlet midlet;
    private String text;
    public HelloCanvas(HelloCanvasMIDlet midlet, String text) {
        this.midlet = midlet;
        this.text = text;

        addCommand(exitCommand);
        setCommandListener(this);
    }
    protected void paint(Graphics g) {
        // Limpa a janela
        g.setColor(255, 255, 255 );
        g.fillRect(0, 0, getWidth(), getHeight());
        // Define a cor como preta
        g.setColor(0, 0, 0);
        // E escreve o texto
        g.drawString(text,
            getWidth()/2, getHeight()/2,
            Graphics.TOP | Graphics.HCENTER);
    }
    public void commandAction(Command c, Displayable d) {
        if (c == exitCommand){
            midlet.Quit();
        }
    }
}
```

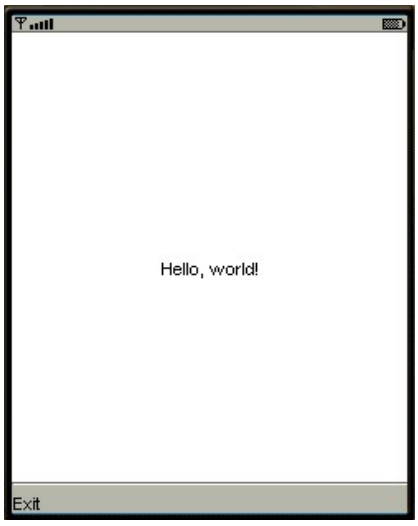


*Figura 3: Hello Word utilizando Canvas*

Com o *midlet* "Hello, world!" definimos a classe que estende a classe *Canvas*

```
class HelloCanvas extends Canvas implements CommandListener {
```

Adicionamos um comando "Exit" (saída) e o colocamos como *listener* do comando

```
addCommand(exitCommand);
setCommandListener(this);
```

Criamos um comando de *listener* através da implementação de uma classe *CommandListener*. Isto significa criar uma classe que tenha um método *commandAction*.

```
class HelloCanvas extends Canvas implements CommandListener {
  public void commandAction(Command c, Displayable d) {
    ...
  }
}
```

O ponto central deste programa é o método *paint*(). A primeira chamada deste método limpa a tela:

```
g.setColor(255, 255, 255 );
g.fillRect(0, 0, getWidth(), getHeight());
```

E os métodos gráficos de baixo nível desenham a mensagem "Hello, world!" na tela:

```
g.setColor(0, 0, 0);
g.drawString(text, getWidth()/2, getHeight()/2,
    Graphics.TOP | Graphics.HCENTER);
```

## *2.3. Comandos*

Assim como as classes *List*, *TextBox* e *Form*, a classe *Canvas* pode possuir *Commands* anexados e pode escutar por eventos de comando. Os passos para adicionar um comando a um *Canvas* são os mesmos:

1. Criar um objeto da classe *Command*

```
private Command exitCommand = new Command("Exit", Command.EXIT, 0);
```

2. Utilizar o método *addCommand*() para anexar o comando ao objeto da classe *Canvas* (*Form*, *List* ou *TextBox*)

```
addCommand(exitCommand);
```

3. Utilizar o método *setCommandListener*() para registrar qual classe obtém os eventos de comando neste *Displayable*

```
setCommandListener(this);
```

4. Criar um *commandListener* para implementar a classe *CommandListener* e providenciar o método *commandAction*()

```
class HelloCanvas extends Canvas implements CommandListener {
    ...
    public void commandAction(Command c, Displayable d) {
        if (c == exitCommand){
            // do something
        }
    }
}
```

## 2.4. Eventos de Teclado

A classe *Canvas* pode ter uma interface *listener* para eventos de teclado que podem implementar os seguintes métodos:

| | |
|---|---|
| `keyPressed(int keyCode)` | Quando uma tecla é pressionada |
| `keyRepeated(int keyCode)` | Quando uma tecla é repetida (quando é mantida pressionada) |
| `keyReleased(int keyCode)` | Quando uma tecla é liberada |

A classe *Canvas* define os seguintes códigos de teclado: KEY_NUM0, KEY_NUM1, KEY_NUM2, KEY_NUM3, KEY_NUM4, KEY_NUM5, KEY_NUM6, KEY_NUM7, KEY_NUM8, KEY_NUM9, KEY_STAR, e KEY_POUND.

Para recuperar o nome da tecla deve-se utilizar o método *getKeyName*(int keyCode).

## 2.5. Ações de Jogo

Cada código de tecla pode ser mapeada para uma ação de jogo. A classe *Canvas* define estas ações: UP, DOWN, LEFT, RIGHT, FIRE, GAME_A, GAME_B, GAME_C, GAME_D. O programa pode traduzir um código de tecla em uma ação de jogo utilizando o método *getGameAction*(int keyCode).

O método *getKeyCode*(int gameAction) retorna o código de tecla associado com uma ação de jogo. Uma ação de jogo pode ter mais de um código de teclado associado. Se houver mais de um código de tecla associado com uma ação de jogo é retornado apenas um único código.

Uma aplicação deve utilizar o método *getGameAction*(int keyCode) ao invés de usar diretamente os códigos de teclado. Normalmente, se um programa precisa esperar por uma tecla "UP", ele deveria utilizar o atributo KEY_NUM2 ou o código específico para o botão UP. No entanto, programas que utilizam este método não são portáveis em aparelhos que possam ter diferentes formatos para o código da tecla do botão UP. O KEY_NUM2 pode ser a "tecla UP" para um aparelho, entretanto, pode ser a "tecla LEFT" em outro aparelho. O método *getGameAction*() deve sempre retornar UP, independente de qual tecla foi pressionada já que é esta que está no contexto do aparelho.

## 2.6. Eventos de apontador

Apesar dos eventos de teclado, programas *MIDP* também podem conter eventos de apontador. Isto é verdade se o aparelho possuir um apontador e estiver implementado no sistema Java do aparelho

O método *hasPointerEvents*() retorna true se o aparelho suportar eventos de pressão e liberar um apontador. O método *hasPointerMotionEvents*() retorna true se o aparelho suportar eventos de movimentação de apontador.

```
public boolean hasPointerEvents()
public boolean hasPointerMotionEvents()
```

Os eventos que são gerados por atividade de ponto são definidos pela implementação dos

métodos: *pointerPressed*, *pointerReleased* e *pointerDragged*. Uma aplicação precisa implementar estes métodos para ser notificada quando ocorrerem. As coordenadas x e y do evento (onde o apontador foi pressionado, solto ou arrastado) são especificadas como parâmetros nas assinaturas dos seguintes métodos:

```
protected void pointerPressed(int x, int y)
protected void pointerReleased(int x, int y)
protected void pointerDragged(int x, int y)
```

# 3. Gráficos

A classe *Graphics* é a principal classe para desenhar textos, imagens, linhas, retângulos e arcos. Possui métodos para especificar cor, fonte e estilo.

## *3.1. Cores*

A classe *Display* possui métodos para determinar se o aparelho possui suporte a cor e o número de cores ou tons de cinza suportados por ele.

| | |
|---|---|
| `public boolean` **`isColor`**`()` | Retorna true se a tela suportar cor, retorna false se não |
| `public int` **`numColors`**`()` | Retorna o número de cores (ou tons de cinza se o aparelho não suportar cores) suportados pelo aparelho |

Para definir a cor que será utilizada para os próximos métodos, utilize o método *setColor*(). Pode ser utilizado de duas maneiras:

```
public void setColor(int red, int green, int blue)
public void setColor(int RGB)
```

Na primeira forma, especifica-se os componentes em padrão RGB, intensidade das cores vermelho (***R**ed*), verde (***G**reen*) e azul (***B**lue*) que podem variar de 0 a 255. Na segunda forma, aos componentes da cor são especificados na forma de 0x00RRGGBB. As chamadas para inteiros no método *setColor* seguem o código da mesma maneira:

```
int red, green, blue;
...
setColor(red, green, blue);
setColor( (red<<16) | (green<<8) | blue );
```

Outro métodos de manipulação de cores são:

| | |
|---|---|
| `public int` **`getColor`**`()` | retornar um inteiro da cor atual do formulário no formato `0x00RRGGBB` |
| `public int` **`getRedComponent`**`()` | retornar o valor da cor vermelho do componente atual |
| `public int` **`getGreenComponent`**`()` | retornar o valor da cor verde do componente atual |
| `public int` **`getBlueComponent`**`()` | retornar o valor da cor azul do componente atual |
| `public int` **`getGrayScale`**`()` | retornar o valor da escala de cinza da cor atual |
| `public void` **`setGrayScale`**`(`<br>`  int value)` | definir o valor da escala de cinza para uma próxima instrução |

## *3.2. Fontes*

Um objeto do tipo *Font* possui três atributos: *face*, *style* e *size*. As fontes não são criadas pela aplicação. Ao invés disso, a aplicação pergunta ao sistema por determinados atributos da fonte e o sistema retorna uma fonte com estes atributos. O sistema não garante que irá retornar uma fonte com todos os atributos requeridos. Se o sistema não tem uma fonte igual a que requerida, retornará uma fonte próxima sem respeitar todos os atributos requeridos.

*Font* é uma classe separada. Como comentado anteriormente, a aplicação não cria um objeto padrão. Ao invés disso os métodos estáticos *getFont*() e *getDefaultFont*() são utilizados para obter uma fonte padrão para o sistema.

| | |
|---|---|
| `public static Font` **`getFont`**`(`<br>`  int face, int style, int size)` | retornar um objeto do tipo *Font* do sistema com os atributos definidos |
| `public static Font` **`getDefaultFont`**`()` | retornar a fonte padrão utilizada pelo sistema |
| `public static Font`<br>**`getFont`**`(int fontSpecifier)` | retornar a fonte utilizada por um componente gráfico de auto nível. O atributo *fontSpecifier* |

| | |
|---|---|
| | pode ser:<br>`FONT_INPUT_TEXT` ou `FONT_STATIC_TEXT` |

O formato da fonte é especificado pelo parâmetro *face* que pode assumir uma das seguintes constantes padrão: FACE_SYSTEM, FACE_MONOSPACE ou FACE_PROPORTIONAL.

O estilo da fonte é especificado pelo parâmetro *style* que pode assumir uma das seguintes constantes padrão: STYLE_PLAIN (padrão) ou a combinação de STYLE_BOLD (negrito), STYLE_ITALIC (itálico) e STYLE_UNDERLINED (sublinhado). Estilos podem ser combinados utilizando o operador OR (|). O estilo de uma fonte tipo negrito e itálico é declarado como:

```
STYLE_BOLD| STYLE_ITALIC
```

O tamanho da fonte é especificado pelo parâmetro *size* que pode assumir uma das seguintes constantes padrão: SIZE_SMALL, SIZE_MEDIUM ou SIZE_LARGE.

Estes métodos retornam os atributos específicos da fonte:

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

## 3.3. Estilo do Traço

O método *setStrokeStyle*(int style) especifica o estilo do cursor que deverá ser usado para desenhar linhas, arcos e retângulos com cantos arredondados. O Estilo do cursor não afeta o texto, as imagens ou os preenchimentos.

| | |
|---|---|
| `public void setStrokeStyle( int style)` | definir o estilo do cursor que deverá ser usado para desenhar linhas, arcos e retângulos com cantos arredondados |
| `public int getStrokeStyle()` | retornar o estilo atual do cursor |

Os valores válidos para o parâmetro *style* são uma das seguintes constantes padrão: SOLID ou DOTTED.

## 3.4. Clipping

*Clipping* é uma área retangular em um objeto gráfico. Qualquer operação gráfica deverá afetar apenas os pontos contidos na área de corte. Pontos fora da área de corte não serão afetados por qualquer operação gráfica.

| | |
|---|---|
| `public void setClip( int x, int y, int width, int height)` | definir a área de corte para o retângulo especificado pelas coordenadas |
| `public int getClipX()` | retornar o valor X da área de corte atual, relativo à origem deste contexto gráfico |
| `public int getClipY()` | retornar o Y da área de corte atual, relativo à origem deste contexto gráfico |
| `public int getClipWidth()` | retornar a largura da área de corte atual |
| `public int getClipHeight()` | retornar a altura da área de corte atual |

## 3.5. Pontos Âncora

Textos são desenhados por um ponto inicial. O método *drawString*() espera as coordenadas localizada nos parâmetros x e y que são relativos aos pontos iniciais.

```
public void drawString(String str, int x, int y, int anchor)
```

O ponto deve ser uma combinação da constante horizontal (LEFT, HCENTER ou RIGHT) e a constante vertical (TOP, BASELINE ou BOTTOM). As constantes horizontal e vertical devem ser combinadas utilizando o operador OU (|), por exemplo para desenhar o texto pela base horizontal do centro, será necessário um valor inicial de BASELINE | HCENTER.

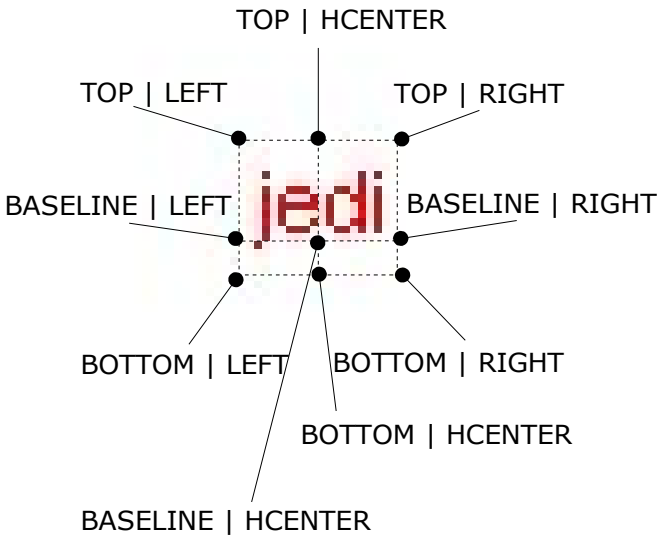


*Figura 4: Texto dos pontos iniciais*

## *3.6. Desenhando Textos*

Os métodos para desenhar texto e caracteres são:

| | |
|---|---|
| `public void drawString( String str, int x, int y, int anchor)` | desenhar o texto definido pelo parâmetro *str* utilizando a cor e fonte atual. Os parâmetros *x* e *y* são as coordenadas do ponto inicial |
| `public void drawSubstring( String str, int offset, int len, int x, int y, int anchor)` | desenhar de maneira semelhante ao método *drawString*, exceto pelos parâmetros *offset* que limita a base-zero e *len* que é responsável pelo tamanho |
| `public void drawChar( char character, int x, int y, int anchor)` | desenhar o caractere usando a cor da fonte atual |
| `public void drawChars( char[] data, int offset, int length, int x, int y, int anchor)` | desenhar os caracteres contidos no parâmetro *data*, iniciando pelo índice definido no parâmetro *offset* com o tamanho especificado em *length* |

Aqui estão alguns métodos da classe *Font* que são úteis para o desenho de texto:

| | |
|---|---|
| `public int` **`getHeight`**`()` | retornar a altura do texto desta fonte. O tamanho retornado inclui os espaçamentos extras. Isto assegura que o desenho de dois textos com esta distância a partir de um ponto de âncora para outro ponto de âncora conterá espaço suficiente entre as duas linhas de texto |
| `public int` **`stringWidth`**`(String str)` | retornar a largura total em pixels do espaço ocupado pelo parâmetro *str* se for escrito utilizando-se esta fonte |
| `public int` **`charWidth`**`(char ch)` | retornar a largura total em pixels do espaço ocupado pelo parâmetro *ch* se for escrito utilizando-se esta fonte |
| `public int` **`getBaselinePosition`**`()` | retornar a distância em pixels entre o TOPO e a BASE do texto, baseado nesta fonte |

```
g.setColor(255, 0, 0); // vermelho
g.drawString("JEDI",
          getWidth()/2, getHeight()/2,
          Graphics.TOP | Graphics.HCENTER);
g.setColor(0, 0, 255); // azul
Font font = g.getFont();
g.drawString("Java Education & Development Initiative",
          getWidth()/2, getHeight()/2+font.getHeight(),
          Graphics.TOP | Graphics.HCENTER);
```

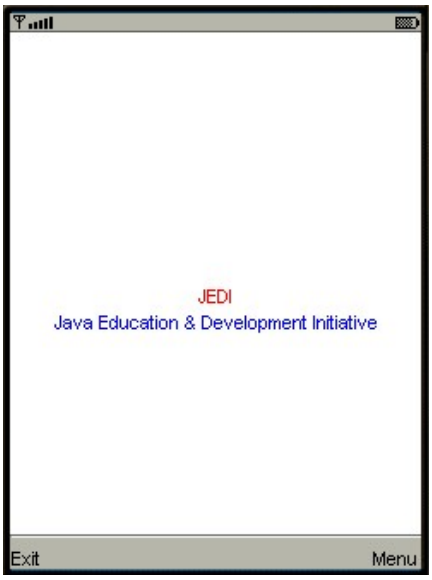


*Figura 5: Saída de operações de drawString()*

## *3.7. Desenhando Linhas*

O único método da classe *Graphics* utilizado para desenhar linhas é definido como:

```
public void drawLine(int x1, int y1, int x2, int y2)
```

Este método desenha uma linha utilizando a cor e estilo corrente entre a coordenada inicial indicada nos parâmetros **x1** e **y1** e a coordenada final indicada nos parâmetros **x2** e **y2**.

```
g.setColor(255, 0, 0); // vermelho
// linha do canto superior esquerdo para o canto inferior direito da tela
g.drawLine(0, 0, getWidth()-1, getHeight()-1);

g.setColor(0, 255, 0); // verde
// linha horizontal no meio da tela
g.drawLine(0, getHeight()/2, getWidth()-1, getHeight()/2);

g.setColor(0, 0, 255); // azul
```

```
// linha horizontal na parte inferior da tela
g.drawLine(0,  getHeight()-1, getWidth()-1, getHeight()-1);

g.setColor(0, 0, 0); // preta
// linha do canto inferior esquerdo para o canto superior direito da tela
g.drawLine(0, getHeight()-1, getWidth()-1,  0);
```

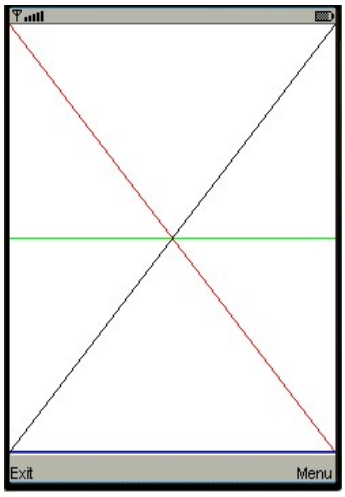


*Figura 6: Saída de chamadas a métodos de drawLine()*

## *3.8. Desenhando Retângulos*

Os métodos da classe *Graphics* para desenhar retângulos são:

```
public void drawRect(int x, int y, int width, int height)
public void drawRoundRect(int x, int y,
                  int width, int height,
                  int arcWidth, int arcHeight)
public void fillRect(int x, int y, int width, int height)
public void fillRoundRect(int x, int y,
                  int width, int height,
                  int arcWidth, int arcHeight)
```

O método *drawRect* desenha um retângulo com o canto superior esquerdo nas coordenadas definidas pelos parâmetros x e y e com área definida em *width*+1 e *height*+1. Os mesmos parâmetros estão no método *drawRoundRect*. Os parâmetros adicionais *arcWidth* e *arcHeight* são os diâmetros horizontal e vertical do arco dos quatro cantos.

Note que a definição dos métodos *drawRect* e *drawRoundRect* especificam que o tamanho do retângulo desenhado na tela é *width*+1 e a altura é *height*+1. Isto não é muito intuitivo, entretanto é desta forma que estes métodos foram definidos para as especificações de *MIDP*.

Para agravar esta inconsistência de adicionar 1 ao tamanho real, os métodos *fillRect* e *fillRoundRect* preenchem um retângulo de área especificada por *width* e *height*. Devido a esta inconsistência pode-se colocar os mesmos parâmetros para os métodos *drawRect, fillRect, drawRoundRect* e *fillRoundRect*. As bordas da direita e da parte inferior do retângulo desenhado pelo método *drawRect* estarão além da área preenchida por *fillRect*.

```
// use tinta preta para drawRect
g.setColor(0, 0, 0);
g.drawRect(8, 8, 64, 32);

// use tinta amarela para fillRect
// para mostrar a diferença entre drawRect e fillRect
g.setColor(255, 255, 0);
g.fillRect(8, 8, 64, 32);
```

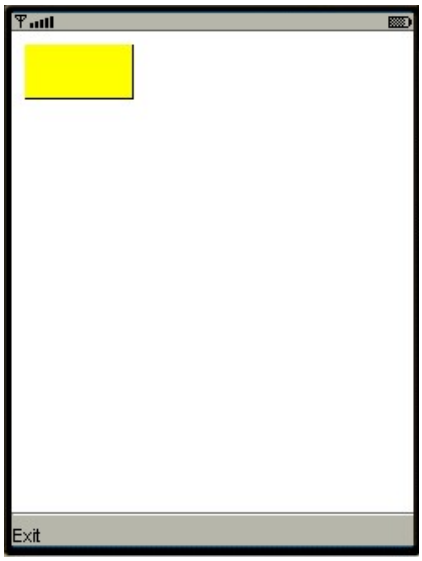


*Figura 7: Saída usando os mesmos parâmetros para drawRect e fillRect*

```
// Define a cor da caneta para preto
g.setColor(0, 0, 0);

// Desenha um retângulo na posição inicial 4 e 8 com largura 88 e altura 44
// o retângulo superior esquerdo
g.drawRect(4,8,88,44);

// Desenha um retângulo arredondado no canto superior direito
g.drawRoundRect(108,8,88,44,18,18);

// Desenha um retângulo no canto inferior esquerdo
g.fillRect(4,58,88,44);

// Desenha um retângulo no canto inferior direito
g.fillRoundRect(108,58,88,44,18,18);
```

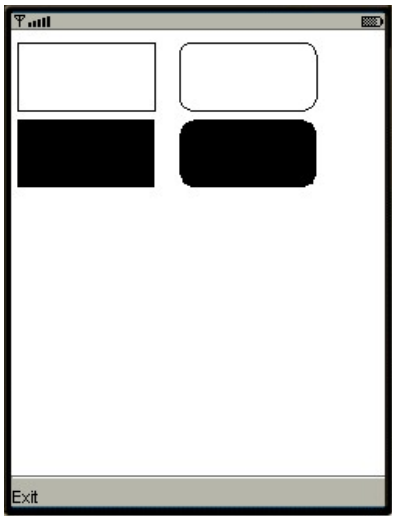


*Figura 8: drawRect(), fillRect(), drawRoundRect() e fillRoundRect()*

## *3.9. Desenhando Arcos*

Os métodos para desenhar arcos circulares ou elípticos são:

| | |
|---|---|
| `public void drawArc( int x, int y, int largura, int altura, int anguloInic, int anguloArco)` | desenhar um arco com centro em (x,y) e dimensões (largura+1 x altura+1). O arco desenhado inicia-se em anguloInic e se estende por anguloArco graus. 0 (zero) grau está na posição 3 horas |
| `public void fillArc( int x, int y, int largura, int altura, int anguloInic, int anguloArco)` | desenhar um um arco circular ou elíptico com toda a sua área coberta com a cor atual |

```
g.setColor(255, 0, 0);
g.drawArc(18, 18, 50, 50, 0, 360); // desenha um círculo
g.setColor(0,  255, 0);
g.drawArc(40, 40, 100, 120, 0, 180);
g.setColor(0, 0, 255);
g.fillArc(100, 200, 80, 100,  0, 90);
```

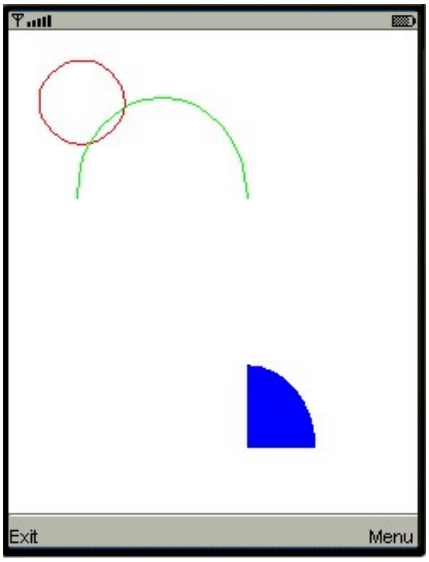


*Figura 9: Saída da chamada aos métodos drawArc e fillArc*

## *3.10. Desenhando imagens*

Imagens são desenhadas usando o método *drawImage*() que possui a seguinte assinatura:

```
public void drawImage(Image img, int x, int y, int anchor)
```

**Atenção**: No NetBeans as imagens devem ser colocadas na pasta **src** do projeto.

Do mesmo modo que no método *drawString*(), os parâmetros x e y são as coordenadas do ponto de âncora. A diferença é que a constante vertical da âncora é VCENTER em vez de BASELINE.

A âncora deve ser uma combinação de uma constante horizontal (LEFT, HCENTER ou RIGHT) e de uma vertical (TOP, VCENTER ou BOTTOM). As constantes horizontal e vertical devem ser combinadas usando o operador de bit OR (|). Isto significa que desenhar texto relativo ao centro vertical e ao centro horizontal da imagem requer uma âncora de valor VCENTER | HCENTER.

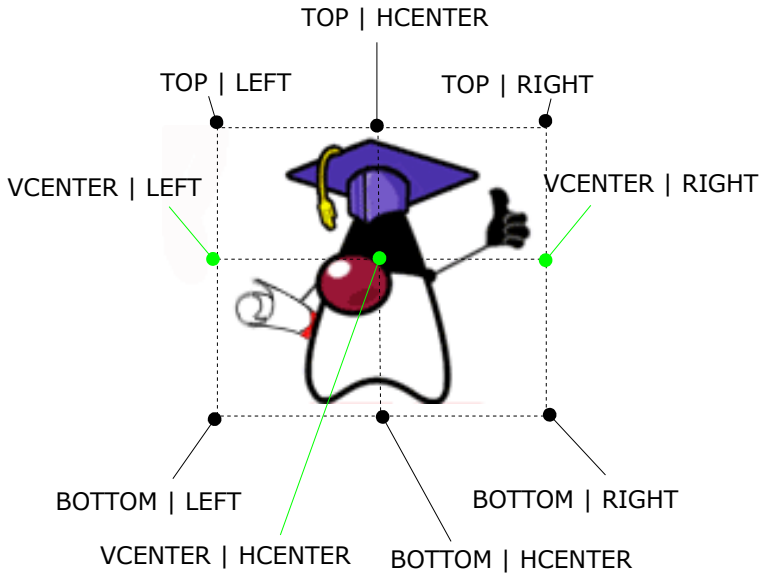


*Figura 10: Pontos de âncora da imagem*

```
try {
    Image image = Image.createImage("/jedi.png");
    g.drawImage(image,
        getWidth()/2, getHeight()/2,
        Graphics.VCENTER | Graphics.HCENTER);
} catch (Exception e){}
```

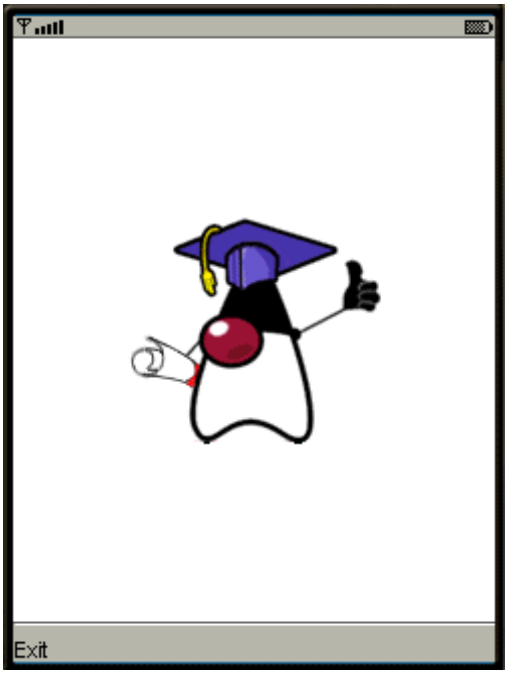


*Figura 11: Saída do método drawImage()*

## 3.11. Programa Final

Iremos combinar as idéias aprendidas nesta lição para obtermos o logo do JEDI na tela do celular.

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

public class LogoMidlet extends MIDlet {
  private Display display;
  private LogoJedi canvas;
```

```
    public void startApp() {
      if (display == null){
        canvas = new LogoJedi(this);
        display = Display.getDisplay(this);
      }
      display.setCurrent(canvas);
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    protected void Quit(){
      destroyApp(true);
      notifyDestroyed();
    }
  }
  class LogoJedi extends Canvas implements CommandListener {
    private Command exitCommand = new Command("Exit", Command.EXIT, 0);
    private LogoMidlet midlet;
    private String text;
    public LogoJedi(LogoMidlet midlet) {
      this.midlet = midlet;
      addCommand(exitCommand);
      setCommandListener(this);
    }
    protected void paint(Graphics g) {
      // Limpa a janela
      g.setColor(255, 255, 255 );
      g.fillRect(0, 0, getWidth(), getHeight());
      // Imagem
      try {
        Image image = Image.createImage("/jedi.png");
        g.drawImage(image,
          getWidth()/2, 80,
          Graphics.VCENTER | Graphics.HCENTER);
      } catch (Exception e){ }
      // Texto
      g.setColor(255, 0, 0); // vermelho
      Font f =
        Font.getFont(Font.FACE_PROPORTIONAL, Font.STYLE_BOLD, Font.SIZE_LARGE);
      g.setFont(f);
      g.drawString("JEDI", getWidth()/2, getHeight()/2,
        Graphics.TOP | Graphics.HCENTER);
      int tamFont = f.getHeight();

      g.setColor(0, 0, 255); // azul
      g.drawLine(20, getHeight()/2+tamFont, getWidth()-20, getHeight()/2+tamFont);
      f = Font.getFont(Font.FACE_PROPORTIONAL, Font.STYLE_PLAIN, Font.SIZE_SMALL);
      g.setFont(f);
      g.drawString("Java Education & Development Initiative",
        getWidth()/2, getHeight()/2+tamFont+3,
        Graphics.TOP | Graphics.HCENTER);
    }
    public void commandAction(Command c, Displayable d) {
      if (c == exitCommand){
        midlet.Quit();
      }
    }
  }
```

# 4. Exercícios

## *4.1. Códigos de Teclas*

Criar um *MIDlet* que mostre o código e o nome da tecla que foi pressionada pelo usuário. Utilizar um *Canvas* para mostrar este código e o nome no centro da tela.

# Parceiros que tornaram JEDI™ possível


***Instituto CTS***
Patrocinador do DFJUG.

***Sun Microsystems***
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

***Java Research and Development Center da Universidade das Filipinas***
Criador da Iniciativa JEDI™.

***DFJUG***
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

***Banco do Brasil***
Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

***Politec***
Suporte e apoio financeiro e logístico a todo o processo.

***Borland***
Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

***Instituto Gaudium/CNBB***
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.