



# Programação Gráfica – Parte 2

Versão em Java – 2006 – PUCPR – Tutoria de Jogos – 1º Ano

---

Paulo V. W. Radtke

[pvwradtke@gmail.com](mailto:pvwradtke@gmail.com)

<http://www.ppgia.pucpr.br/~radtke/jogos/>



# AVISO IMPORTANTE!!

---

- Esta versão é dedicada exclusivamente para o cursos de **Sistemas de Informação**.
- Para a versão de **Ciência da Computação e Engenharia da Computação**, utilizando **C**, pegue o arquivo correspondente e participe da aula no horário adequado.



# Entrega da 2ª Parcial

---

- Datas importantes:
  - 12 a 17 de Junho.
  - Relatório impresso contendo:
    - Código fonte do protótipo da interface.
    - Impressão no relatório dos recursos gráficos da fase/jogo (*tilemaps*, sprites, cenários de fundos, protótipo, etc).
    - Discussão do uso dos recursos com a lógica do jogo no terceiro bimestre.
  - Defesa em laboratório do protótipo e entrega do relatório com a equipe completa.



# Entrega da 2ª Parcial

---

- Por recursos gráficos, entende-se que neste bimestre já teremos:
  1. TODOS os sprites necessários para o demo.
  2. TODAS as telas de fundo/tilemaps.
  3. TODAS as fontes.
- Logo, espera-se que o relatório inclua TODOS estes elementos.



# Tarefas de Casa

---

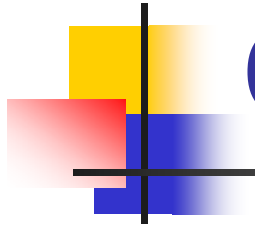
- Alguém leu a documentação do J2ME?
- Se sim, você viu como é implementado o loop principal de um jogo?



# Conteúdo

---

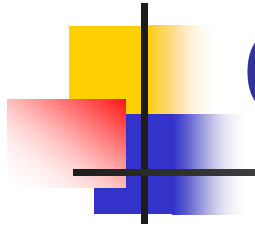
- O *loop* principal do jogo.
- Temporização via *threads*.
- Lendo o teclado.



## O *Loop* Principal

---

- Na aula anterior, vimos como fazer um **GameCanvas** desenhar na tela uma string.
- Este método é interessante para aplicações tradicionais, que usam os componentes de interface tradicionais do J2ME.



## O *Loop* Principal

---

- O problema deste método é que a tela é redesenhada apenas quando o celular determina que ela deve ser redesenhada.
- Para um jogo, o ideal é que de tempos em tempos a tela seja redesenhada.
- Esse “de tempos em tempos” de preferência deve ser um intervalo fixo.





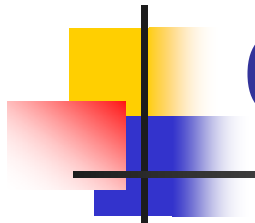
# O *Loop* Principal

---

- A documentação do GameCanvas dá o exemplo de um loop principal de jogo:

```
// Pega o contexto gráfico
Graphics g = getGraphics();
while (true)
{
    // Pinta a tela de branco
    g.setColor(0xFFFFFFFF);
    g.fillRect(0,0,getWidth(), getHeight());
    // Roda a lógica e desenha gráficos

    // Sincroniza buffers de desenho
    flushGraphics();
}
```



## O *Loop* Principal

---

- Agora falta uma maneira de colocar este código dentro de um método Java que seja chamado de tempos em tempos.
- Para fazermos isto, utilizaremos uma *Thread*.



# Temporização Via *Threads*

---

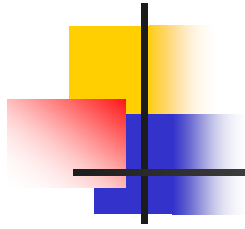
- Uma *thread* consiste em uma linha de execução de código independente da aplicação, rodando em paralelo a esta.
- Logo, a aplicação principal fica esperando eventos, enquanto a *thread* prossegue rodando.
- Assim, podemos usar uma *thread* para implementar o *loop* principal do jogo.



# Temporização Via *Threads*

---

- Uma thread é obtida em java através da interface *Runnable*.
- Todo objeto que implemente esta interface deve possuir um método **run**.
- Este método é chamado uma única vez, roda em paralelo com a aplicação e quando encerra, termina a *thread*.



## *GameCanvas com Thread*

---

- Geralmente, o próprio objeto que possui a thread faz o processo de inicialização.
- No nosso caso, a inicialização é feita no **GameCanvas**.
- O Midlet então indica para o GameCanvas quando a thread deve começar e terminar, de acordo com as chamadas a **startApp** e **pauseApp**.



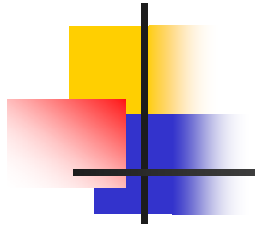
# *GameCanvas com Thread*

---

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

public class JogoCanvas extends GameCanvas implements Runnable
{
    private int maxX; // Largura da tela
    private int maxY; // Altura da tela
    // Atributo para indicar estado do jogo
    private boolean rodando = false;

    public JogoCanvas()
    {
        super(true);
        maxX = getWidth();
        maxY = getHeight();
    }
}
```

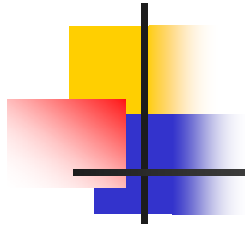


# *GameCanvas com Thread*

---

```
public void inicia()
{
    // Indica que deve rodar
    rodando=true;
    // Cria a thread baseado neste objeto
    Thread t = new Thread(this);
    // Inicia a thread
    t.start();
}

public void pausa()
{
    // Indica que a thread deve encerrar
    rodando=false;
}
```



# *GameCanvas com Thread*

---

```
// Este método é chamado a pelo objeto Thread do Java.
public void run()
{
    // Se for pra rodar
    while(rodando)
    {
        Graphics g = getGraphics();
        g.setColor(255, 255, 255);
        g.fillRect(0, 0, maxX, maxY);
        g.setColor(0, 0, 0);
        g.drawString("Rodando em thread.",
maxX/2,maxY/2,Graphics.HCENTER | Graphics.BASELINE);
        // Sincroniza o reedesenho
        flushGraphics();
    }
}
}
```





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

```
public class BaseJogo extends MIDlet  
{  
    private Display display;  
    private JogoCanvas canvas;  
  
    public BaseJogo()  
    {  
        display = Display.getDisplay(this);  
        canvas = new JogoCanvas();  
    }  
}
```



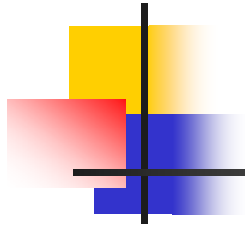
## *Midlet*

```
public void startApp()
{
    display.setCurrent(canvas);
    canvas.inicia();
}
```

```
public void pauseApp()
{
    canvas.pausa();
}
```



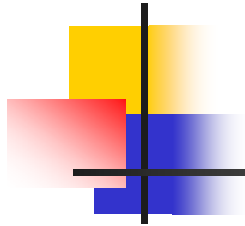
```
public void destroyApp(boolean unconditional)
{
    canvas.pausa();
}
}
```



## Exemplo 01

---

- Crie um projeto novo no KToolBar e adicione os arquivos do exemplo.
- O arquivo é: **j2me-exemplo02.zip**
- Perguntas a se fazer:
  - Qual o nome do projeto?
  - Qual o nome do midlet?



# Exercício 01

---

- Aparentemente não há diferenças deste Midlet para o HelloWorld da aula passada.
- Crie um contador no GameCanvas que conte quantas vezes a repetição do while foi executada.
- Imprima o contador junto com o texto mostrado na tela.
- Observe o resultado.



## Pausas em *Threads*

---

- A thread desta maneira está rodando direto, sem pausas.
- Para que a thread rode de tempos em tempos, como requerido por um jogo, precisamos fazer com que esta pare de executar por alguns instantes.



## Pausas em *Threads*

---

- Isto é obtido com o `Thread.sleep` .
- Este método estático faz com que a Thread “durma” e depois do tempo especificado em milissegundos ela volta a rodar.
- Este método lança uma exceção. Na CNTP, é interessante tratar o erro, mas em geral não costuma falhar.



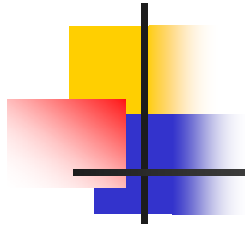
## Pausas em *Threads*

---

- Uso típico (pausa de 50ms):

```
try
{
    Thread.sleep(50);
}
catch(InterruptedException e){
```

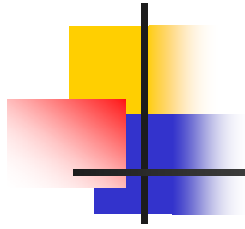




## Exercício 02

---

- Modifique o exercício anterior para que, além de contar o número de repetições, esta repetição seja executada 10 vezes por segundo.



# Entrada do Teclado

---

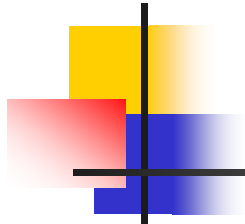
- O *GameCanvas* tem associado a ele o teclado do aparelho.
- Pelo menos, a parte relevante para um jogo:
  - Direcionais.
  - Um botão de tiro.
  - Quatro botões (A a D), **de acordo com o aparelho** (não padrão).



# Entrada do Teclado

---

- O acesso ao teclado é feito através do método **getKeyStates** do GameCanvas.
- Este método retorna o estado destas teclas, que pode ser pressionado ou não pressionado.
- O método retorna UM inteiro combinando a informação completa.



# Entrada do Teclado

---

- Os estados individuais são obtidos através de uma combinação bit a bit do valor retornado com um identificador da tecla.

- Ex:

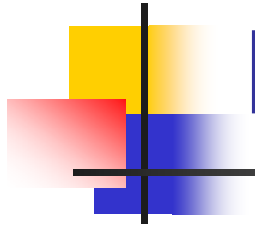
```
int tecla=getKeyStates();  
if(tecla & UP_PRESSED)  
{  
    // Pressionou a tecla pra cima
```



# Entrada do Teclado

---

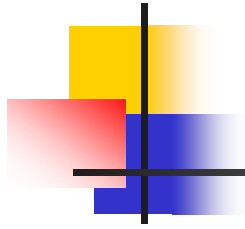
- O código para verificar o estado destas teclas é simples e será repetido em diversas aplicações.
- O mais adequado é colocar um atributo booleano para cada tecla trabalhada, indicando o seu estado.
- Depois, um método, chamado pela lógica, atualiza estes atributos.



# Entrada do Teclado

---

```
public void leTeclas()
{
    int teclas = getKeyStates();
    // TEstas as teclas
    if((teclas & LEFT_PRESSED)!=0)
        esquerda=true;
    else
        esquerda=false;
    if((teclas & RIGHT_PRESSED)!=0)
        direita=true;
    else
        direita=false;
```



# Entrada do Teclado

---

```
    if((teclas & UP_PRESSED)!=0)
        cima=true;
    else
        cima=false;
    if((teclas & DOWN_PRESSED)!=0)
        baixo=true;
    else
        baixo=false;
    if((teclas & FIRE_PRESSED)!=0)
        tiro=true;
    else
        tiro=false;
}
```



## Exemplo 02

---

- Crie um novo projeto para o exemplo no arquivo **j2me-exemplo03.zip**.
- Rode o projeto e veja o resultado.
- Analise o código.

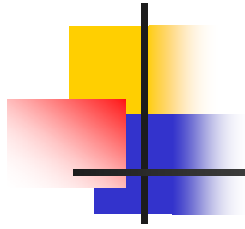




# Próxima Aula

---

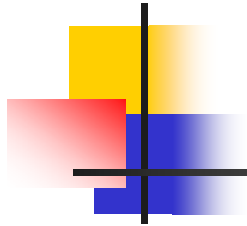
- O GameCanvas em detalhes.
- Desenhando um sprite na tela.
- Movendo o sprite com eventos do teclado.



## Exercício 03

---

- Modifique o exercício anterior para fazer um texto se mover pela tela.
  1. Crie as coordenadas **x** e **y** deste texto.
  2. Quando as teclas forem pressionadas, atualize o valor de **x** e **y**.
  3. Desenhe o texto na posição indicada por **x** e **y**.



# Próxima Aula

---

- Desenhando uma Imagem
- Desenhando um Sprite
- Desenhando um TileMap