CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
Campus Contagem
Atividade Prática - Programação em Java

SÉRIE: 3ª CURSO: INFORMÁTICA

1 – Desenvolva as atividades e envie as fontes em um arquivo zipado com seu nome.
Tutorial de criação de jogo em Java- Parte 1:

A Janela Principal
Iniciaremos criando a janela principal do jogo. Para tal, utilizaremos o componente JPanel (javax.swing.JPanel) do Swing.

Utilizando o editor de sua preferência, inicie um novo arquivo e coloque o nome de **JogoPainel.java**

Insira o seguinte código nele:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JogoPainel extends JPanel {
  private static final int LARGURA = 640;
  private static final int ALTURA = 480;

  public JogoPainel() {
    setBackground(Color.BLACK);
    setPreferredSize(new Dimension(LARGURA, ALTURA));
  }
}
```

Comentários sobre o código:

0 - O código não faz nada, somente define as características básicas da janela.

1. Includes necessários para uso dos componentes utilizados no jogo

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

2. Cria a classe principal e extende as funcionalidades do JPanel

public class JogoPainel exteds JPanel { ... }

3. Define duas constantes, ALTURA e LARGURA, que são as dimensões da tela do jogo

private static final int LARGURA = 640;

private static final int ALTURA = 480;

4. Cria o construtor da classe

public JogoPainel() { ... }

5. Define as características do painel, que são cor de fundo(background) e tamanho do painel respectivamente

setBackground(Color.BLACK);

setPreferredSize(new Dimension(LARGURA, ALTURA));

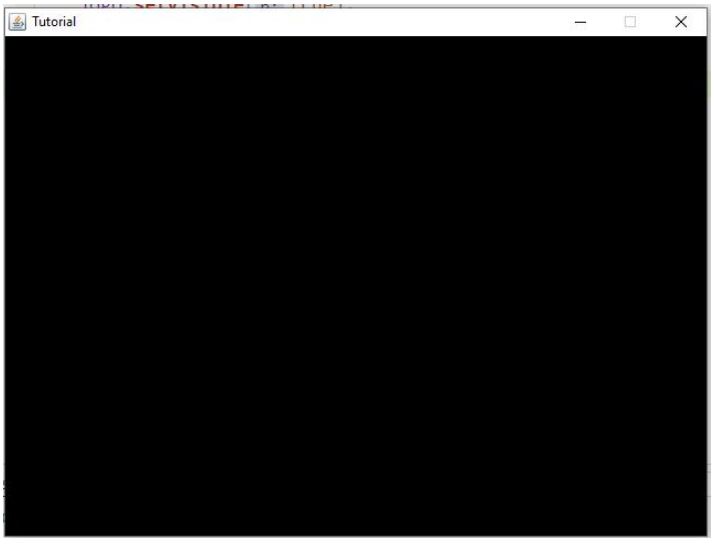
Esse código é o suficiente para exibirmos o painel na tela, porém, para que ele possa ser exibido, é necessário que o painel seja adicionado a um frame ou a um applet no caso de exibir o jogo via navegador.

No nosso caso iremos adicioná-lo a um JFrame, também um componente Swing.

O método main
Crie um novo arquivo com o nome de **Jogo.java** e coloque o seguinte código nele:

```
import javax.swing.JFrame;
public class Jogo {
        public static void main(String args[])
        {
    JogoPainel jogoPainel = new JogoPainel();
            JFrame jogo = new JFrame("Tutorial");
            jogo.add(jogoPainel);
            jogo.pack();
            jogo.setSize(640, 480);
            jogo.setResizable(false);
            jogo.setVisible(true);
            jogo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Rode o arquivo e você verá uma tela preta:



Comentários sobre o código:

1. Includes necessários para uso dos componentes utilizados no jogo

import javax.swing.JFrame;

2. Classe principal do jogo

public class Jogo { ... }

3. Método principal ativado pelo sistema

public static void main(String args[]) { ... }

4. Declaração do painel contendo o jogo

JogoPainel jogoPainel = new JogoPainel();

5. Declaração do Frame(JFrame) onde será adicionado o painel do jogo criado anteriormente

JFrame jogo = new JFrame("Tutorial");

6. Adição do painel do jogo no frame, adaptação do tamanho do painel no frame(pack), tornar frame visível e possibilitar que o frame seje fechado ao clicar no botão fechar da janela respectivamente.

jogo.add(jogoPainel);

jogo.pack();

jogo.setVisible(true);

jogo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

Aqui termina a primeira parte do tutorial, na segunda parte será apresentado como criar um MainLoop utilizando Threads, até lá.

Thread para controlar MainLoop (laço principal) e FPS.

Parte 2:

Iniciaremos agora a parte dois do tutorial, onde será criado o MainLoop para o jogo utilizando Thread, lembrando que esse recurso é utilizado em jogos tipo arcade, onde é necessário uma atualização constante da tela.

Mas o que é Thread?

Thread nada mais é do que processos que rodam concorrentemente com outros processos, assim podemos executar várias ações ao mesmo tempo, dentro do nosso jogo um exemplo seria que enquanto o jogo é exibido na tela, existe um outro processo que estará escutando para verificar se alguma tecla ou botão do mouse foram pressionados. Para melhor entendimento, é aconselhável ler algum tutorial ou livro que aborde Thread em detalhes.

Sabendo o que é Thread, vamos implantá-lo em nosso código.

Obs.: o código abaixo contém algumas ações somente para efeito de testes da Thread implementada, usadas para verificar o comportamento da Thread, não sendo necessário se preocupar com elas no momento.

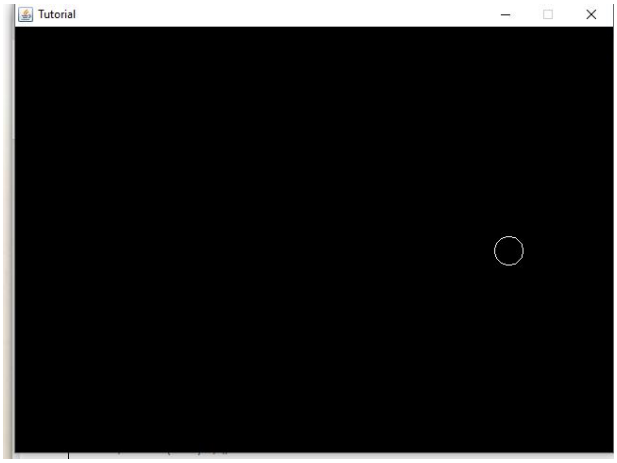
Altere o conteúdo do arquivo **JogoPainel.java** para o código abaixo:

```
import java.awt.*;
import javax.swing.*;
import javax.swing.JPanel;
```

```
public class JogoPainel extends JPanel implements Runnable {
	// Tamanho da janela - tamanho das bordas (640-15 = 625)
	private static final int LARGURA = 625;
	// Tamanho da janela - tamanho das bordas (480-35 = 445)
	private static final int ALTURA = 445;
	private Thread animacao;
	private int tempoEspera = 10;
	private boolean jogando = false;
	int diametroBola = 30;
	/*
	 * variáveis com posição e direção iniciais utilizadas apenas para efeito de
	 * testes
	 */
	private int x = 50, y = 150, velocidadeX = 1, velocidadeY = 1;

	// Define informações da janela.
	public JogoPainel() {
		// cor de fundo.
		setBackground(Color.BLACK);

	}

	// Controle de início do jogo.
	public void addNotify() {
		super.addNotify();
		iniciarJogo();
	}

	// Função que da start nas funções de início do jogo
	private void iniciarJogo() {
		if (animacao == null || !jogando) {
			animacao = new Thread(this);
			animacao.start();
		}
	}

	public void run() {
		jogando = true;
		/*
		 * Loop do jogo. Aqui as coisas acontecem.
		 */
		while (jogando) {
			/*
			 * Coloque aqui todas as ações que queira que sejam executadas a cada loop do
			 * jogo
			 */
			moveBola();
			repaint();
			/* fim ações para testes */
			try {
				Thread.sleep(tempoEspera);
			} catch (InterruptedException e) {
				e.printStackTrace();
```

```
                        }
                }
                System.exit(0);
        }

        /* Função que desenha o fundo do painel . */
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                // Define a cor
                g.setColor(Color.WHITE);
                // Pinta o circulo. (tente g.fillOval)
                g.drawOval(x, y, diametroBola, diametroBola);
        }

        // Função que controla o movimento da bola.
        private void moveBola() {
                // Bora quebra um pouco a cabeça.
                if (x >= LARGURA - diametroBola)
                        velocidadeX *= -1;
                if (x <= 0)
                        velocidadeX *= -1;
                if (y >= ALTURA - diametroBola)
                        velocidadeY *= -1;
                if (y <= 0)
                        velocidadeY *= -1;
                x += velocidadeX;
                y += velocidadeY;
        }
        /* Fim! (Será) */
}
```

Execute a classe Jogo.java e veja uma bolinha se movendo na tela.



Agora será explicado cada linha do código necessária para a implementação da Thread.

1. A classe do jogo implementa a interface Runnable, utilizada pela Thread

public class JogoPainel extends JPanel implements Runnable { ... }

2. É declarada uma variável do tipo Thread, que será responsável por inicializar a Thread do jogo

private Thread animacao;

3. Variável utilizada para determinar o tempo(em milissegundos, ms) de espera entre as ações do jogo

private int tempoEspera = 12;

4. Variável utilizada para controlar o MainLoop, ou seja, se o jogo está ou não em execução

private boolean jogando= false;

5. Função que é chamada automaticamente pelo sistema quando inicializado, será utilizada para dizer a hora em que o painel(JPanel) foi carregado, assim, a Thread pode ser iniciada e o jogo começar. É uma função com o objetivo de garantir que a Thread não inicialize até que o painel do jogo esteja totalmente carregado

public void addNotify() { ... }

6. Adiciona a notificação ao JPanel

super.addNotify();

7. Chamada à função responsável por iniciar a Thread, chamada após a notificação de que o painel do jogo está carregado

iniciarJogo();

8. Como citado acima, é a função que cria e inicializa a Thread do jogo

private void iniciarJogo() { ... }

9. Verifica se a Thread foi criada e se o jogo já iniciou

if(animacao==null || !jogando) { ... }

10. Caso Thread não instanciada, cria uma instância da Thread, com o construtor Thead que recebe como parâmetro uma instância de uma classe que implementa a interface Runnable, nesse caso a própria classe(this)

animacao = new Thread(this);

11. Inicia a Thread

animacao.start();

12. O metodo run() deve ser implementado obrigatoriamente, pois ele faz parte da interface Runnable. è um método muito importante, pois é o primeiro a ser chamado quando a Thread é iniciada(nomeThread.start()), e ele conterá o MainLoop do jogo

public void run() { ... }

13. Seta a variável jogando para verdadeiro, pois o jogo será iniciado

jogando = true;

14. Laço principal(MainLoop) do jogo, todas as ações serão contidas nele, como atualização dos dados, desenho da tela...

while(jogando) { ... }

15. Utilizado para tratamento de exceções

try { ... }

16. Faz com que a Thread aguarde o tempo passado como parâmetro entre as execuções do laço while, no código de exemplo, esse tempo é de 12ms, o que garante aproximadamente 83(1000/12=83.34, onde 1000 ms equivales a 1 segundo) execuções em um segundo, assunto que será discutido mais adiante em FPS(Frames Por Segundo). Experimente alterar esse valor para verificar o que ocorre com o comportamento da Thread

Thread.sleep(tempoEspera);

17. Thread.sleep() lança uma exceção(InterruptedException) caso ocorra um erro, e deve ser tratada com try/catch, que é o que ocorre abaixo, para maiores detalhes, leia a respeito de tratamento de exceções em Java

catch(InterruptedException e) { ... }

18. Imprime no prompt o erro caso lance uma exceção

e.printStackTrace();

19. Sai do jogo quando terminar a execução

System.exit(0);

Interagindo com a janela.

Parte 3:

Agora acrescentamos a parte de interação com a bolinha.

Java trata interação do usuário por meio de ouvintes (listeners). Existem ouvintes para teclado, mouse, ações das janelas, etc.

Vamos modificar um pouco o código de JogoPainel conforme o código abaixo:

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.event.*;
import javax.swing.JPanel;

public class JogoPainel extends JPanel implements Runnable, KeyListener {
	// Tamanho da janela - tamanho das bordas (640-15 = 625)
	private static final int LARGURA = 625;
	// Tamanho da janela - tamanho das bordas (480-35 = 445)
	private static final int ALTURA = 445;
	private Thread animacao;
	private int tempoEspera = 10;
	private boolean jogando = false;
	int diametroBola = 30;
	/*
	 * variáveis com posição e direção iniciais utilizadas apenas para efeito de
	 * testes
	 */
	private int x = 50, y = 150, velocidadeX = 1, velocidadeY = 1;

	// Define informações da janela.
	public JogoPainel() {
		// cor de fundo.
		setBackground(Color.BLACK);
		addKeyListener(this);
		setFocusable(true);
	}

	// Controle de início do jogo.
```

```
	public void addNotify() {
		super.addNotify();
		iniciarJogo();
	}

	// Função que da start nas funções de início do jogo
	private void iniciarJogo() {
		if (animacao == null || !jogando) {
			animacao = new Thread(this);
			animacao.start();
		}
	}

	public void run() {
		jogando = true;
		/*
		 * Loop do jogo. Aqui as coisas acontecem.
		 */
		while (jogando) {
			/*
			 * Coloque aqui todas as ações que queira que sejam executadas a cada loop do
			 * jogo
			 */
			moveBola();
			repaint();
			/* fim ações para testes */
			try {
				Thread.sleep(tempoEspera);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
		System.exit(0);
	}

	/* Função que desenha o fundo do painel . */
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		// Define a cor
		g.setColor(Color.WHITE);
		// Pinta o circulo. (tente g.fillOval)
		g.drawOval(x, y, diametroBola, diametroBola);
	}

	// Função que controla o movimento da bola.
	private void moveBola() {
		// Bora quebra um pouco a cabeça.
		if (x >= LARGURA - diametroBola)
			velocidadeX *= -1;
		if (x <= 0)
			velocidadeX *= -1;
		if (y >= ALTURA - diametroBola)
			velocidadeY *= -1;
```

```
                if (y <= 0)
                        velocidadeY *= -1;
                x += velocidadeX;
                y += velocidadeY;
        }

        /* Fim! (Será) */
        @Override
        public void keyPressed(KeyEvent e) {
                switch (e.getKeyCode()) {
                case KeyEvent.VK_UP:
                        System.out.println("Pressionado up");
                        break;
                case KeyEvent.VK_DOWN:
                        break;
                case KeyEvent.VK_LEFT:
                        break;
                case KeyEvent.VK_RIGHT:
                        break;
                default:
                        break;
                }
        }

        // Quando a tecla for solta
        @Override
        public void keyReleased(KeyEvent e) {
                switch (e.getKeyCode()) {
                case KeyEvent.VK_UP:
                        System.out.println("Solto Up");
                        break;
                case KeyEvent.VK_DOWN:
                        break;
                case KeyEvent.VK_LEFT:
                        break;
                case KeyEvent.VK_RIGHT:
                        break;
                default:
                        break;
                }
        }

        @Override
        public void keyTyped(KeyEvent e) {
        }
}
```

Observe as mudanças:

1. Incluída a biblioteca de eventos:

import java.awt.event.*;

2. Adicionada na assinatura da classe um KeyListener - escutador de teclas.

public class JogoPainel extends JPanel implements Runnable, KeyListener {

3. Adicionado ao construtor do painel a indicação de que ele tem um escutador de teclas e que ele pode receber o foco do programa.

addKeyListener(this);

setFocusable(true);

4. Os métodos de tratamento do uso das teclas é adicionado:

    Tecla pressionada.

    public void keyPressed(KeyEvent e) {}

    Tecla solta.

    public void keyReleased(KeyEvent e) {}

    Tecla inserida (funciona como a tecla pressionada)

    public void keyTyped(KeyEvent e) {}

Aqui somente verificamos qual tecla foi pressionada e solta e damos um alerta na janela do console.

Como podemos mover a bolinha a partir desse comando?

5. Modifique a linha:

private int x = 50, y = 150, velocidadeX = 1, velocidadeY = 1;

para:

private int x = 50, y = 150, velocidadeX = 0, velocidadeY = 1;

Isso altera quantidade de movimento das direções x e y.

Faça seus testes utilizando valores para x e y como 1 (pra frente) ou -1 (pra trás).

6. Os if's do método moveBola() serão nossos case no metodo keyReleased(KeyEvent e) .

direita é x positivo.

esquerda é x negativo.

para cima é y negativo.

para baixo é y positivo.

Faça as alterações em **1** e **-1** (um e menos um) para as variáveis velocidadeX e velocidadeY e isso já controlará o movimento da bolinha.

7. Remova a chamada ao método moveBola() comentando a linha

    moveBola();

Verifique o que é necessário transferir do método moveBola(); para dentro dos métodos de controle de teclas para que o movimento ocorra de forma mais natural.

8. Agora modifique seu programa para ter duas bolinhas. Uma movimentando sozinha e uma outra movimentando pelo teclado.

Extraindo objetos do código.

Parte 4:

Você deve ter observado a dificuldade em gerar as duas bolinhas na tela. Imagine então criar 50. Para esse tipo de tarefa POO vem em grande auxílio.

Pensando no tutorial 3 verifique o todo o código que foi duplicado para inserção da nova bolinha. Esse código deve ser extraído e colocado em uma nova classe.

1. Crie uma classe chamada Movel.
2. Transfira para essa classe todas as variáveis que pertencem somente a bolinha (somente uma vez, não a necessidade de repetir para construir duas ou mais bolinhas).
3. Insira na classe Movel os métodos de controle das bolinhas com seus comandos.

   public void moveBola(){} -> mude o nome somente para move();

   public void paintComponent(Graphics g)
4. Você deve decidir se cria um construtor para definir todas as variáveis do móvel logo no início ou cria métodos para alterá-los após serem inicializados.

Passando os dados no construtor uma sugestão é a seguinte assinatura:

public Movel(int novoX, int novoY, int novaAltura, int novaLargura, Color novaCor, int novoTamanho) {...}

Ou você pode utilizar um construtor vazio e passar as informações por meio de métodos set ou um método de inicialização de parâmetros que você vier a definir.

5. No construtor da classe JogoPainel crie um objeto da classe Movel em uma variável de nome bola e defina os seus valores iniciais.
6. Na classe JogoPainel onde é chamado o método moveBola() mude para bola.move();

Isso deve funcionar.

7. Na classe JogoPainel dentro da função paintComponent troque o comando
   g.drawOval(x, y, diametroBola, diametroBola);
   por uma chamada de paintComponent do movel.

   bola.paintComponent(g);

8. Execute o código e verifique o bom funcionamento do programa agora usando uma classe para organizar os objetos em movimento.
9. Agora crie 10 objetos da classe Movel com posições, cores e tamanhos diferentes.

   Para isso use a classe ArrayList para controlar as bolas.

   Use for's (loops) sobre coleções para acessar os elementos da lista.

Referências sobre ArrayList

http://www.javaprogressivo.net/2012/09/como-usar-arraylist-em-java.html

http://www.devmedia.com.br/explorando-a-classe-arraylist-no-java/24298