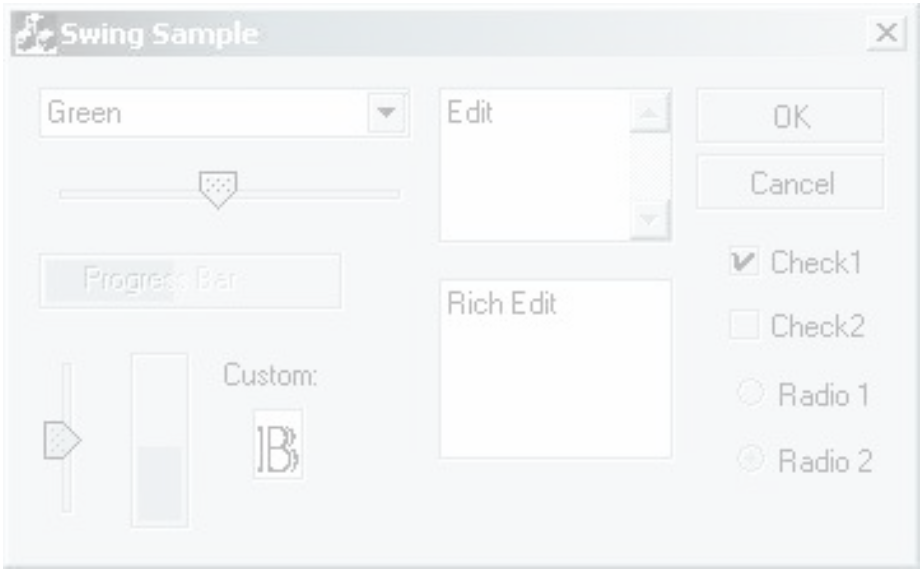


Aula 18

Componentes GUI - Parte II

Rafael Geraldeli Rossi

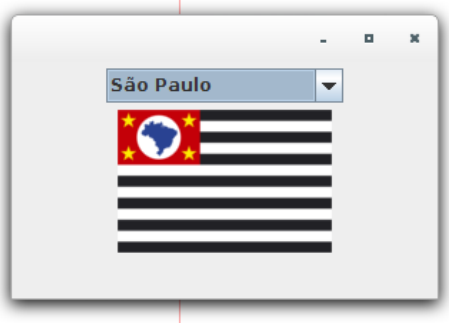
JComboBox

- Um `JComboBox`, ou **caixa de combinação**, ou **lista drop-down**, permite ao usuário selecionar um item de uma lista
- Porém, a lista é retraída, isto é, o usuário não enxerga todas as opções da lista a menos que ele clique no `JComboBox` para que este apresente a lista completa
- Os `JComboBoxes` geram `ItemEvents`

- Para adicionar elementos em um JComboBox, pode-se passar um array de objetos via construtor ou um utilizar o método addItem();
- Como um JComboBox recebe como parâmetro Objetos para adicionar, diferentes tipos de elementos podem ser renderizados
- Para se recuperar o índice do elemento selecionado usa-se o método `getSelectedIndex()`, e para recuperar o conteúdo do elemento selecionado usa-se o método `getSelectedItem()`

JComboBox

```
19  public class Janela extends JFrame{
20
21      private JComboBox combo;
22      private JLabel labelImg;
23      private String[] nomesLista;
24      private Icon[] icones;
25
26      Janela(){
27          this.setLayout(new FlowLayout());
28          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29          this.setSize(300, 200);
30
31          nomesLista = new String[4];
32          nomesLista[0] = ""; nomesLista[1] = "São Paulo"; nomesLista[2] = "Bahia"; nomesLista[3] = "Mato Grosso do Sul";
33          icones = new Icon[4];
34          icones[0] = new ImageIcon("");
35          icones[1] = new ImageIcon("/home/rafael/Imagens/bandeira_sao_paulo.png");
36          icones[2] = new ImageIcon("/home/rafael/Imagens/bandeira_bahia.png");
37          icones[3] = new ImageIcon("/home/rafael/Imagens/bandeira_mato_grosso_do_sul.png");
38
39          combo = new JComboBox(nomesLista);
40          ItemHandler handler = new ItemHandler();
41          combo.addItemListener(handler);
42          this.add(combo);
43
44          labelImg = new JLabel();
45          this.add(labelImg);
46
47          this.setVisible(true);
48      }
49
50      private class ItemHandler implements ItemListener{
51
52          public void itemStateChanged(ItemEvent e) {
53              labelImg.setIcon(icones[combo.getSelectedIndex()]);
54          }
55      }
56  }
```



JComboBox

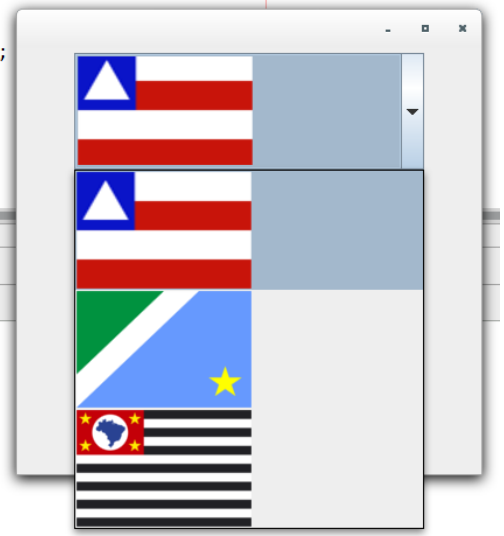
```
public class TesteJanela extends JFrame{

    JLabel lLogar;
    JTextField tLogar;
    JButton bLogar;
    JComboBox cEstados;
    Icon[] icones;

    public TesteJanela(){

        this.setSize(400, 400);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new FlowLayout());

        icones = new ImageIcon[3];
        icones[0] = new ImageIcon("/home/rafael/Teste7/bandeira_bahia.png");
        icones[1] = new ImageIcon("/home/rafael/Teste7/bandeira_mato_grosso_do_sul.png");
        icones[2] = new ImageIcon("/home/rafael/Teste7/bandeira_sao_paulo.png");

        cEstados = new JComboBox(icones);
        cEstados.setPreferredSize(new Dimension(300,100));

        this.add(cEstados);

        this.setVisible(true);

    }
```

teJanela ▸ TesteJanela ▸

Resultados da Pesquisa ×

onsole do Depurador × | Teste (run) ×

run:



JList

- Um `JList`, ou simplesmente uma lista, exibe uma série de itens a partir da qual o usuário pode selecionar um ou mais itens
- A Classe `JList` suporta **listas de uma única seleção** (que permitem que apenas um item seja selecionado por vez) e **listas de seleção múltipla** (que permite que qualquer número de itens seja selecionado)
- Para definir o modo de seleção da lista deve-se utilizar o método `setSelectionMode(int valor)`, no qual pode-se utilizar campos estáticos da classe `ListSelectionModel` para informar o valor

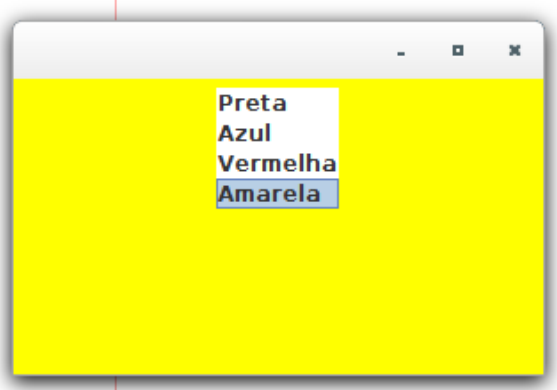
JList

- Para se recuperar os índices ou valores dos itens selecionados pode-se utilizar os métodos:
  - `getSelectedIndex()`
  - `getSelectedIndices()`
  - `getSelectedValue()`
  - `getSelectedValues()`

- Quando se clica em um item de uma lista ocorre um `ListSelectionEvent`

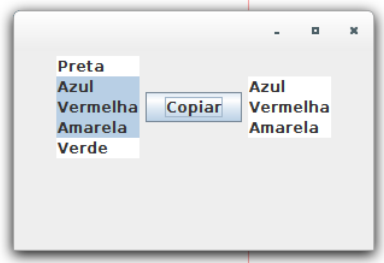
Lista de Seleção Simples

```
27      Janela(){
            janela = this;
            this.setLayout(new FlowLayout());
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setSize(300, 200);
32
33          nomesCores = new String[4];
34          nomesCores[0] = "Preta"; nomesCores[1] = "Azul"; nomesCores[2] = "Vermelha"; nomesCores[3] = "Amarela";
35          cores = new Color[4];
36          cores[0] = Color.BLACK; cores[1] = Color.BLUE; cores[2] = Color.RED; cores[3] = Color.YELLOW;
37
38          lista = new JList(nomesCores);
39          lista.setVisibleRowCount(5);
40          lista.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
            this.add(lista);
42
43          lista.addListSelectionListener(new ListHandler());
44
            this.setVisible(true);
46      }
47
48      private class ListHandler implements ListSelectionListener{
49
            public void valueChanged(ListSelectionEvent e) {
51              janela.getContentPane().setBackground(cores[lista.getSelectedIndex()]);
52          }
53
54      }
55  }
```



Lista de Seleção Múltipla

```
18  public class Janela extends JFrame{
19
20      private JList listaOriginal;
21      private JList listaCopia;
22      private String[] nomesCores;
23      private JFrame janela;
24      private JButton botao;
25
26      Janela(){
27          janela = this;
28          this.setLayout(new FlowLayout());
29          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30          this.setSize(300, 200);
31
32          nomesCores = new String[5];
33          nomesCores[0] = "Preta"; nomesCores[1] = "Azul"; nomesCores[2] = "Vermelha";
34          nomesCores[3] = "Amarela"; nomesCores[4] = "Verde";
35
36          listaOriginal = new JList(nomesCores);
37          listaOriginal.setVisibleRowCount(5);
38          listaOriginal.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
39          this.add(listaOriginal);
40
41          botao = new JButton("Copiar");
42          this.add(botao);
43          botao.addActionListener(new BotaoHandler());
44
45          listaCopia = new JList();
46          listaCopia.setVisibleRowCount(5);
47
48          this.add(listaCopia);
49
50          this.setVisible(true);
51      }
52
53
54      private class BotaoHandler implements ActionListener{
55
56          @Override
57          public void actionPerformed(ActionEvent e) {
58              listaCopia.setListData(listaOriginal.getSelectedValues());
59          }
60
61      }
62
63  }
```



Lista de Seleção Múltipla

- Em um `JList` pode-se listar elementos de diferentes tipos

```
    JList lLista;
    Icon[] icones;

    public TesteJanela(){

        this.setSize(400, 400);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new FlowLayout());

        icones = new ImageIcon[3];
        icones[0] = new ImageIcon("/home/rafael/Teste7/bandeira_bahia.png");
        icones[1] = new ImageIcon("/home/rafael/Teste7/bandeira_mato_grosso_do_sul.png");
        icones[2] = new ImageIcon("/home/rafael/Teste7/bandeira_sao_paulo.png");

        lLista = new JList(icones);
        this.add(lLista);


        this.setVisible(true);

    }

}
```

Tratamento de Eventos de Mouse

- Para se tratar eventos de mouse, pode-se utilizar as interfaces `MouseListener` e `MouseMotionListener`
- Os eventos de mouse podem ser capturados por qualquer componente GUI que é derivado de `java.awt.Component`
- O pacote `javax.swing` contém a interface `MouseInputListener`, que estende as interfaces `MouseListener` e `MouseMotionListener` para criar uma única interface que contém todos os métodos de `MouseListener` e `MouseMotionListener`

Métodos das Interface MouseListener e MouseMotionListener

Métodos da Interface `MouseListener`

Método	Descrição
`mousePressed`	Chamado quando um botão do mouse é pressionado e enquanto o cursos do mouse estiver sobre um componente
`mouseClicked`	Chamado quando um botão do mouse é pressionado e liberado, e enquanto o cursos do mouse estiver sobre um componente
`mouseRelease`	Chamado quando um botão do mouse é liberado depois de pressionado
`mouseEntered`	Chamado quando o cursor do mouse entra nos limites de um componente
`mouseExited`	Chamado quando o cursor do mouse deixa os limites de um componente

Métodos das Interface MouseListener e MouseMotionListener

Métodos da Interface `MouseMotionListener`

Método	Descrição
`mouseDragged`	Chamando enquanto: (i) o botão do mouse é pressionado; e (ii) o cursor do mouse estiver sobre um componente; e (iii) enquanto o botão do mouse permanecer pressionado.
`mouseMoved`	Chamado enquanto o mouse é movido sobre um componente mas sem que algum botão esteja pressionado

Tratamento de Eventos de Mouse

- Cada um dos métodos de tratamento de eventos de mouse recebe como um argumento um objeto `MouseEvent`, que contém informações sobre o evento de mouse que ocorreu, incluindo as coordenadas x e y da localização em que ocorreu o evento
- As coordenadas x iniciam em 0 e aumentam da esquerda para a direita e as coordenadas y iniciam em 0 e aumentam de cima para baixo
- Os métodos e as constantes da classe `InputEvent` (superclasse de `MouseEvent`) permitem-lhe determinar o botão do mouse em que o usuário clicou

Interface `MouseWheelListener`

- O Java também fornece a interface `MouseWheelListener` para permitir que aplicativos respondam à rotação da roda de um mouse
- Essa interface declara o método `mouseWheelMoved` que recebe um `MouseWheelEvent` como seu argumento
- A classe `MouseWheelEvent` (uma subclasse de `MouseEvent`) contém métodos que permitem que o `handler` de evento obtenha as informações sobre a quantidade de rotações da roda

Capturando Eventos do Mouse

```
23          pPainel.setBackground(Color.red);
24          pPainel.addMouseListener(new MouseListenerHandler());
25          pPainel.addMouseMotionListener(new MouseMotionHandler());
            this.add(pPainel,BorderLayout.CENTER);
27
28          lStatus = new JLabel("Aqui");
            this.add(lStatus,BorderLayout.SOUTH);
30
            this.setVisible(true);
32
33      }
34
35      private class MouseListenerHandler implements MouseListener{
36          @Override
            public void mouseClicked(MouseEvent e) {
38              lStatus.setText("Clicou em [" + e.getX() + "," + e.getY() + "]");
39          }
40          @Override
            public void mousePressed(MouseEvent e) {
42              lStatus.setText("Pressionou em [" + e.getX() + "," + e.getY() + "]");
43          }
44          @Override
            public void mouseReleased(MouseEvent e) {
46              lStatus.setText("Liberou em [" + e.getX() + "," + e.getY() + "]");
47          }
48          @Override
            public void mouseEntered(MouseEvent e) {
50              lStatus.setText("Entrou em [" + e.getX() + "," + e.getY() + "]");
51          }
52          @Override
            public void mouseExited(MouseEvent e) {
54              lStatus.setText("Saiu em [" + e.getX() + "," + e.getY() + "]");
55          }
56      }
57
58      private class MouseMotionHandler implements MouseMotionListener{
59          @Override
            public void mouseDragged(MouseEvent e) {
61              lStatus.setText("Clicou e arrastou em [" + e.getX() + "," + e.getY() + "]");
62          }
63          @Override
            public void mouseMoved(MouseEvent e) {
65              lStatus.setText("Movido em [" + e.getX() + "," + e.getY() + "]");
66          }
```

Classes Adaptadoras

- Uma **classe adaptadora** implementa uma interface e fornece uma implementação padrão (com o corpo de um método vazio) de cada método na interface
- Você pode estender uma classe adaptadora para herdar a implementação padrão de cada método e, subsequentemente, sobrescrever somente o(s) método(s) necessário(s) para o tratamento dos eventos
- Uma **classe adaptadora** implementa uma interface e fornece uma implementação padrão (com o corpo de um método vazio) de cada método na interface

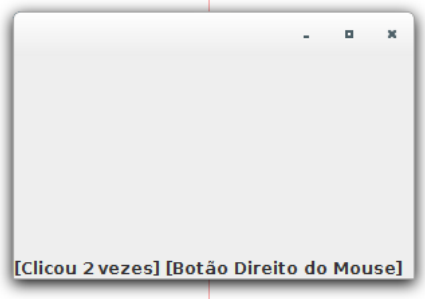
Classes Adaptadoras

- Você pode estender uma classe adaptadora para herdar a implementação padrão de cada método e, subsequentemente, sobrescrever somente o(s) método(s) necessário(s) para o tratamento dos eventos

Classe Adaptadora (`java.awt.event`)	**Interface Implementada**
`ComponentAdapter`	`ComponentListener`
`ContainerAdapter`	`ContainerListener`
`FocusAdapter`	`FocusListener`
`KeyAdapter`	`KeyListener`
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`WindowAdapter`	`WindowListener`

Classes Adaptadoras

```
16  public class Janela extends JFrame{
17
18      private JLabel status;
19
20      Janela(){
21          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22          this.setSize(300, 200);
23
24          status = new JLabel();
25          this.add(status, BorderLayout.SOUTH);
26
27          this.addMouseListener(new MouseClickHandler());
28
29          this.setVisible(true);
30      }
31
32
33      private class MouseClickHandler extends MouseAdapter{
34
35          public void mouseClicked(MouseEvent e) {
36              String saida = "";
37
38              int xPos = e.getX();
39              int yPos = e.getY();
40
41              saida = "[Clicou " + e.getClickCount() + " vezes] ";
42
43              if(e.isMetaDown()){
44                  saida += "[Botão Direito do Mouse] ";
45              }else if(e.isAltDown()){
46                  saida += "[Botão do Meio do Mouse] ";
47              }else{
48                  saida += "[Botão Esquerdo do Mouse] ";
49              }
50
51              status.setText(saida);
52          }
53
54      }
55
56  }
```



Tratamento de Eventos do Teclado

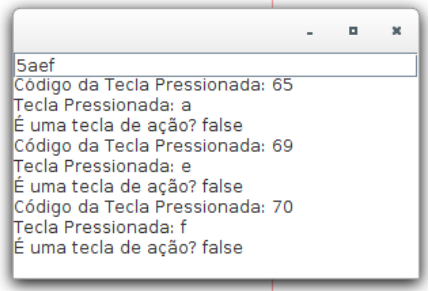
- Eventos de teclado são gerados quando as teclas no teclado são pressionadas e liberadas
- Para tratar eventos de teclado devemos utilizar a interface `KeyListener`
- Uma classe que implemente `KeyListener` deve fornecer declarações para métodos `keyPressed`, `keyReleased` e `keyTyped` $\rightarrow$ todos recebem um `keyEvent` como seu argumento

Tratamento de Eventos do Teclado

- `keyPressed`: chamado em resposta ao pressionamento de qualquer tecla
- `keyTyped`: é chamado em resposta ao pressionamento de qualquer tecla que não seja uma **telca de ação** (setas direcionais, `Home`, `End`, `Page Up`, `Page Down`, teclas de funções, ...)
- `keyReleased`: é chamado quando a tecla é liberada (ocorre depois dos eventos `keyPressed` e `keyTyped`)

Tratamento de Eventos do Teclado

```
18  public class Janela extends JFrame{
19
22      private JTextField texto;
23      private JTextArea areaTexto;
26
27      Janela(){
29          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30          this.setSize(300, 200);
31
32          texto = new JTextField(10);
33          this.add(texto, BorderLayout.NORTH);
34          texto.addKeyListener(new TecladoHandler());
35
36          areaTexto = new JTextArea();
37          areaTexto.setColumns(10);
38          areaTexto.setRows(10);
39          areaTexto.setEditable(false);
40          areaTexto.setLineWrap(true);
41          this.add(areaTexto, BorderLayout.SOUTH);
42
43          this.setVisible(true);
44      }
45
46      private class TecladoHandler implements KeyListener{
47          @Override
48          public void keyTyped(KeyEvent e) {
49              areaTexto.append("Tecla Pressionada: " + e.getKeyChar() + "\n");
50          }
51
52          @Override
53          public void keyPressed(KeyEvent e) {
54              areaTexto.append("Código da Tecla Pressionada: " + e.getKeyCode() + "\n");
55          }
56
57          @Override
58          public void keyReleased(KeyEvent e) {
59              areaTexto.append("É uma tecla de ação? " + e.isActionKey() + "\n");
60          }
61      }
62  }
```

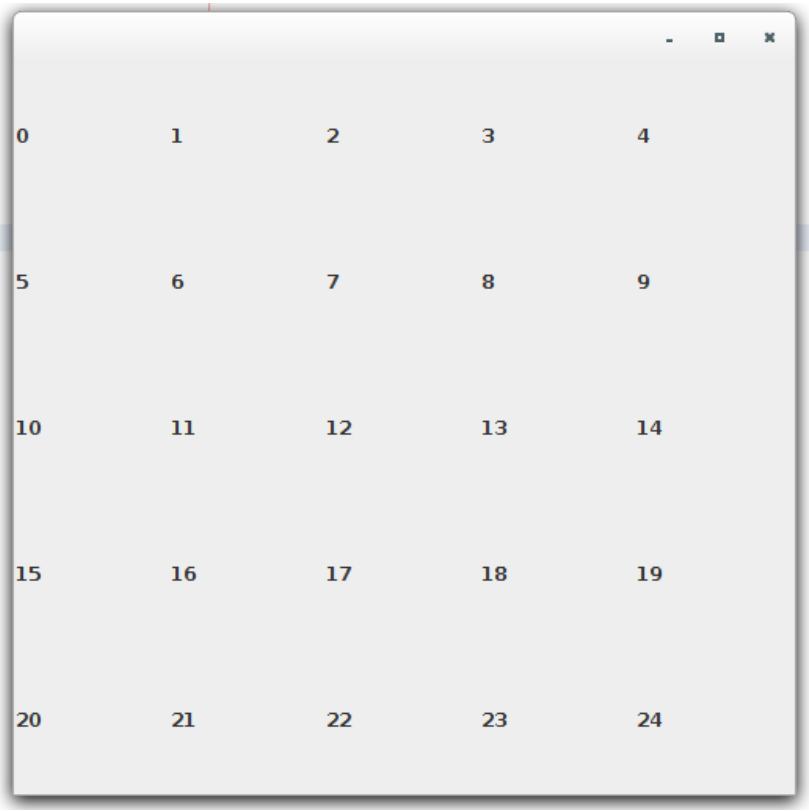


GridLayout

- Para iniciarmos o movimento da nossa cobrinha pela tela, vamos utilizar o gerenciador de layout `GridLayout`
- O `GridLayout` divide o contêiner em uma grade de modo que os componentes podem ser colocados nas linhas e colunas
- Cada `Component` em um `GridLayout` tem a mesma largura e altura
- Os componentes são adicionados a um `GridLayout` iniciando a célula na parte superior esquerda da grade e prosseguindo da esquerda para a direita até a linha estar cheia

GridLayout

```
17
18
19  public class Janela extends JFrame{
20
21      Janela(){
            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setSize(500, 500);
24
            GridLayout gLayout = new GridLayout(5,5);
            this.setLayout(gLayout);
27
28          for(int i=0;i<25;i++){
                this.add(new JLabel(String.valueOf(i)));
30          }
31
            this.setVisible(true);
33      }
34
35
36  }
37
38
```

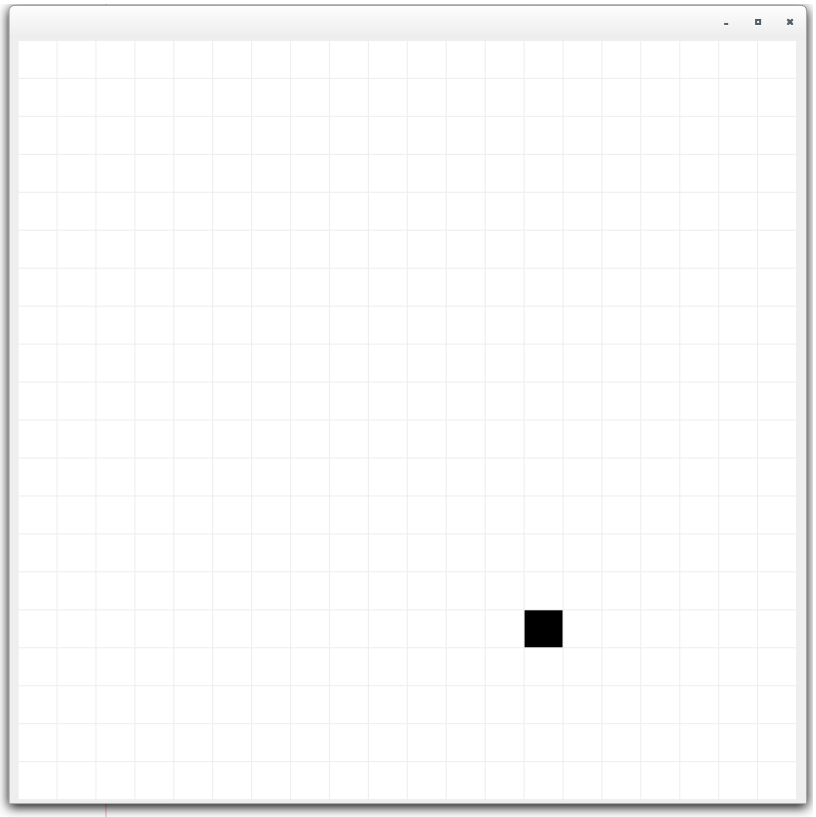


Movendo apenas um ponto no grid

- Vamos começar com uma versão mais básica na qual queremos apenas mover um ponto na tela conforme o usuário pressiona as teclas de direção
- Utilizaremos a variável `xAtual` para controlar a posição do ponto no eixo vertical e a variável `yAtual` para controlar a posição do ponto na vertical
- Utilizaremos painéis para renderizar os pontos do jogo

Movendo apenas um ponto no grid

```
        this.setSize(800, 800);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GridLayout gLayout = new GridLayout(20,20,1,1);
        this.setLayout(gLayout);
        random = new Random();
        panels = new JPanel[20][20];
        for(int i=0;i<20;i++){
            for(int j=0;j<20;j++){
                panels[i][j] = new JPanel();
                panels[i][j].setBackground(Color.white);
                this.add(panels[i][j]);
            }
        }

        xAtual = random.nextInt(20);
        yAtual = random.nextInt(20);
        panels[yAtual][xAtual].setBackground(Color.black);

        this.addKeyListener(new KeyHandler());
        this.setVisible(true);
    }

    private class KeyHandler extends KeyAdapter{
        @Override
        public void keyPressed(KeyEvent e) {
            int xNovo = xAtual;
            int yNovo = yAtual;
            if(e.getKeyCode() == 38){ // Para cima
                yNovo--;
            }else if(e.getKeyCode() == 40){ //Para baixo
                yNovo++;
            }else if(e.getKeyCode() == 39){ // Para direita
                xNovo++;
            }else if(e.getKeyCode() == 37){ // Para esquerda
                xNovo--;
            }
            // Só alterar a cor do quadrado se a posição for válida
            if(xNovo >= 0 && xNovo < 20 && yNovo >= 0 && yNovo <20 ){
                panels[yAtual][xAtual].setBackground(Color.white);
                panels[yNovo][xNovo].setBackground(Color.black);
                xAtual = xNovo;
                yAtual = yNovo;
            }
        }
    }
```



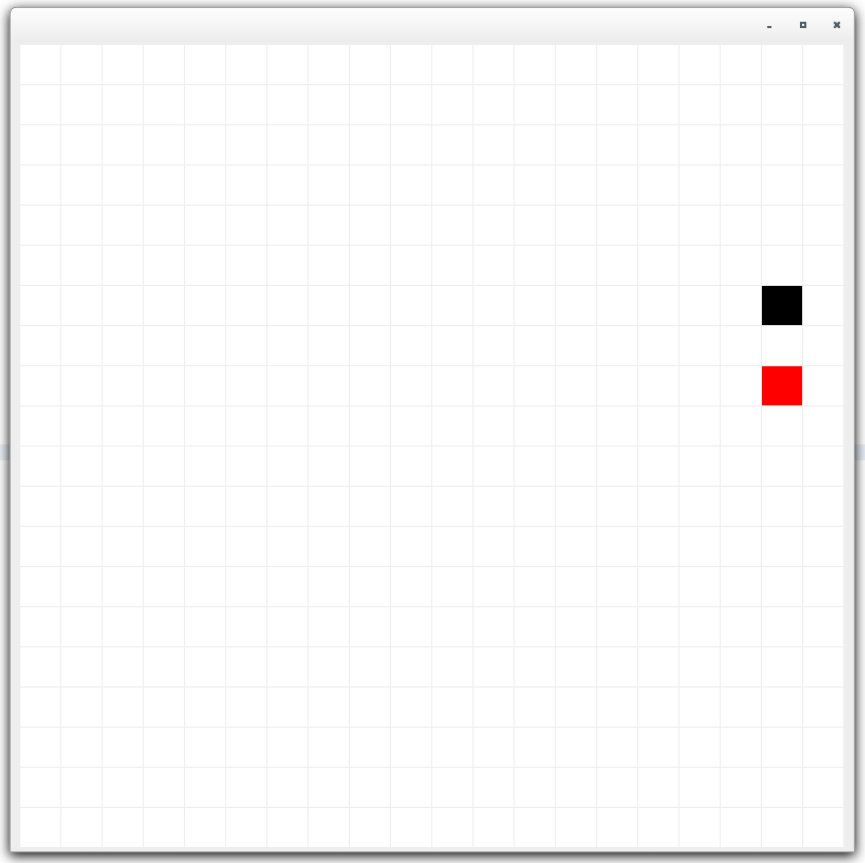
Cobrinha de boa

- Agora vamos estender para fazer com que a posição do ponto seja atualizada em um determinado período de tempo e considerando que o ponto vai ser atualizado de acordo com a última tecla pressionada pelo usuário

- Vamos também fazer aparecer um ponto aleatoriamente na tela ("alvo") o qual será trocado de lugar toda vez que o nosso ponto passar por cima dele

Cobrinha de boa

```
40
41          tecla = "";
42
43          xAtual = random.nextInt(20);
44          yAtual = random.nextInt(20);
45          xAlvo = random.nextInt(20);
46          yAlvo = random.nextInt(20);
47
48          panels[yAtual][xAtual].setBackground(Color.black);
49          panels[yAlvo][xAlvo].setBackground(Color.red);
50
53          this.addKeyListener(new KeyHandler());
54          this.setVisible(true);
55
54          boolean sair = false;
55          while(sair == false){
57              sair = movePonto();
59              try {
60                  Thread.sleep(200);
61              } catch (InterruptedException ex) {
62                  Logger.getLogger(TesteJanela.class.getName()).log(Level.SEVERE, null, ex);
63              }
64          }
63          JOptionPane.showMessageDialog(null, "Acaboooooooooou!");
64          System.exit(0);
65      }
66
67      public boolean movePonto(){
68          int xNovo = xAtual;
69          int yNovo = yAtual;
70
72          if(tecla.equals("cima")){ // Para cima
73              yNovo--;
74          }else if(tecla.equals("baixo")){ //Para baixo
75              yNovo++;
76          }else if(tecla.equals("direita")){ // Para direita
77              xNovo++;
78          }else if(tecla.equals("esquerda")){ // Para esquerda
79              xNovo--;
80          }
81
82          // Só alterar a cor do quadrado se a posição for válida
83          if(xNovo >= 0 && xNovo < 20 && yNovo >= 0 && yNovo <20 ){
84              panels[yAtual][xAtual].setBackground(Color.white);
85              panels[yNovo][xNovo].setBackground(Color.black);
86              xAtual = xNovo;
87              yAtual = yNovo;
88
89              if(xAtual == xAlvo && yAtual == yAlvo){
90                  xAlvo = random.nextInt(20);
91                  yAlvo = random.nextInt(20);
92                  panels[yAlvo][xAlvo].setBackground(Color.red);
93              }
94              return false;
95          }else{
96              return true;
97          }
```

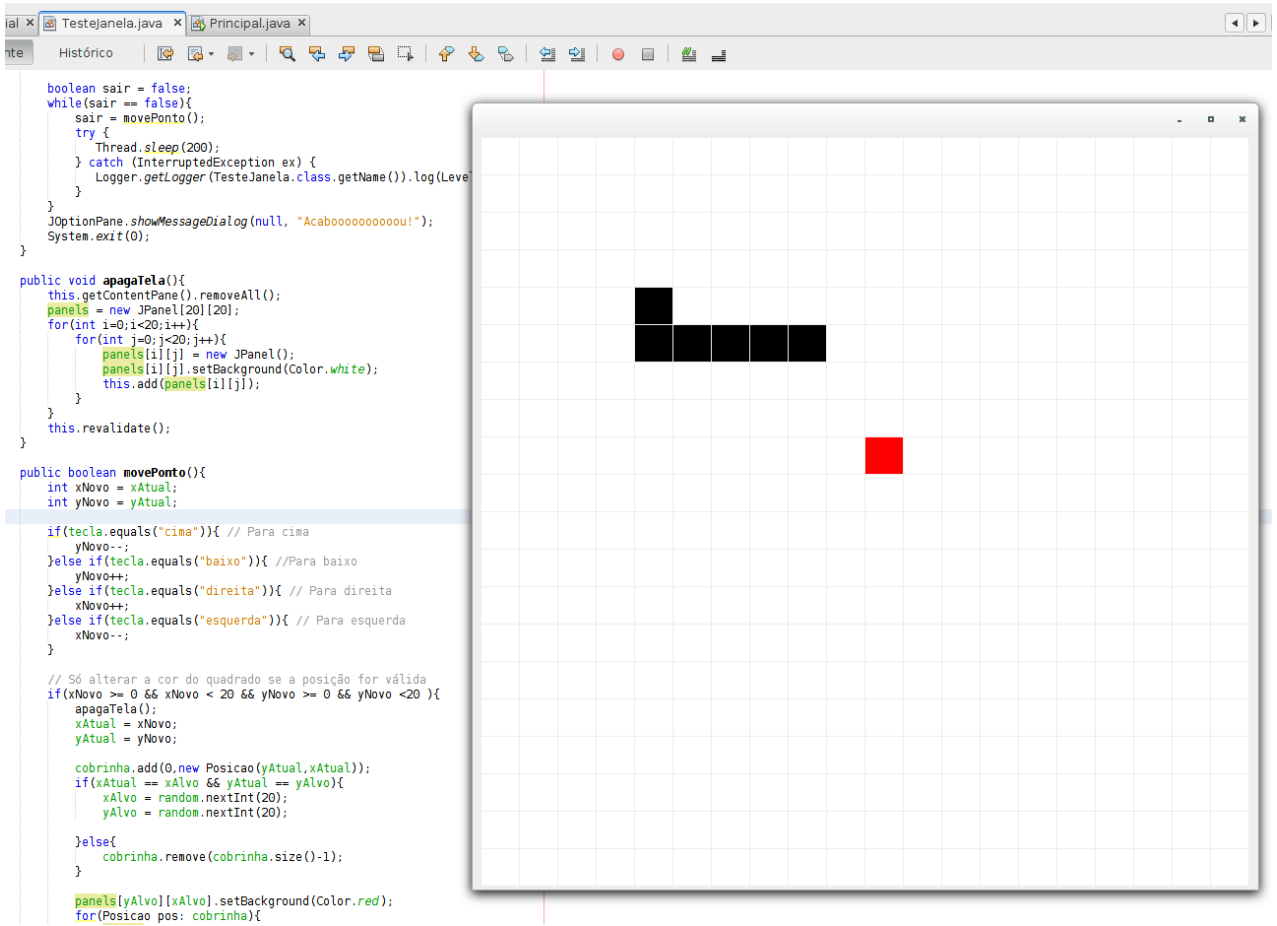


Cobrinha mais profissional

- Podemos agora usar uma estrutura para armazenar todos os pontos da cobrinha e fazê-la aumentar de tamanho ao encontrar o ponto alvo

- Por questões de renderização, vamos criar um novo grid a cada atualização e vamos utilizar o método `revalidate()` para forçar a atualização do grid

Cobrinha mais profissional



Extra: Desenhando em um Panel

- Podemos utilizar um JPanel como uma **área dedicada de desenho** em que o usuário pode desenhar arrastando o mouse

- Os componentes Swing leves que estendem a classe JCompontent (caso do JPanel) contêm o método paintComponent, que é chamado quando um componente Swing leve é exibido

Extra: Desenhando em um Panel

- Ao sobrescrever esse método, pode-se especificar como desenhar formas utilizando capacidades de imagens gráficas do Java

- Ao personalizar um `JPanel` para uso como uma área dedicada de desenho, a subclasse deve sobrescrever o método `paintComponent` e chamar a versão de superclasse de `paintComponent` como a primeira instrução no corpo do método sobrescrito para assegurar que o componente será exibido corretamente

Extra: Desenhando em um Panel

```
2  import java.awt.Graphics;
3  import java.awt.Point;
4  import java.awt.event.MouseEvent;
5  import java.awt.event.MouseMotionAdapter;
6  import java.util.ArrayList;
7  import javax.swing.JPanel;
8
9  public class PainelDesenho extends JPanel{
10
11     private ArrayList<Point> points;
12
13     public PainelDesenho(){
14         points = new ArrayList<Point>();
15         this.addMouseMotionListener(new MouseHandler());
16     }
17
18
19     @Override
20     public void paintComponent(Graphics g){
21         super.paintComponent(g);
22
23         for(int i=0;i<points.size();i++){
24             g.fillOval(points.get(i).x, points.get(i).y, 4, 4);
25         }
26     }
27
28     public class MouseHandler extends MouseMotionAdapter{
29
30         @Override
31         public void mouseDragged(MouseEvent e){
32             points.add(e.getPoint());
33             repaint();
34         }
35     }
36
37 }
```

Extra: Desenhando em um Panel

```
8    */
9
10   /**
11    *
12    * @author rafael
13    */
14   public class Desenho extends JFrame {
15
16       public Desenho() {
17           initComponents();
18           PainelDesenho painel = new PainelDesen
19           painel.setSize(400, 400);
             this.setSize(400, 400);
             this.add(painel);
             this.setTitle("Desenhando");
             this.setVisible(true);
24       }
25
26
27
28
29
30
31       /**
32        * This method is called from within the c
```

Material Complementar

- Curso de Java #05 - Introdução ao Swing e JavaFX

  https://www.youtube.com/watch?v=cYMruFKwqf0

- SWING - Componentes mais importantes e suas propriedades - parte 1

  http://www.devmedia.com.br/

  swing-componentes-mais-importantes-e-suas-propriedades-parte-1/

  16113

Material Complementar

- SWING - Componentes mais importantes e suas propriedades - parte 2

  http://www.devmedia.com.br/swing-componentes-mais-importantes-e-suas-propriedades-parte-2/16258

- Java Swing: Conheça os componentes JTextField e JFormattedTextField http://www.devmedia.com.br/java-swing-conheca-os-componentes-jtextfield-e-jformattedtextfield/30981

Material Complementar

Programação Orientada a Objetos
http://lives.ufms.br/moodle/

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I

Deitel, P. and Deitel, H. (2010).
Java: How to Program.
How to program series. Pearson Prentice Hall, 8th edition.