Interfaces gráficas

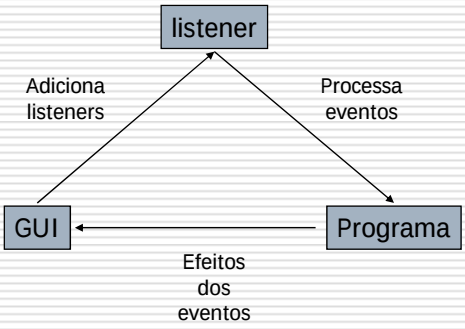
- Muitas aplicações disponibilizam aos seus utilizadores uma interface gráfica (GUI – Graphical User Interface)
- As packages java.awt e javax.swing suportam a criação deste tipo de interfaces
- Os elementos chave de uma interface gráfica em Java são:
  - Componentes
  - Gestores de posicionamento (layout managers)
  - Processadores de eventos
- Componentes são elementos como botões ou campos de texto que podem ser manipulados pelo utilizador com o rato ou o teclado

Interfaces gráficas

- Gestores de posicionamento determinam a forma como os componentes aparecem no ecrã
- Processadores de eventos respondem a acções do utilizadores (ex: pressão num botão do rato)
- Os programas com interfaces gráficas devem responder a eventos, gerados pelos componentes, que indicam que determinadas acções ocorreram
- Existe uma categoria de classes chamada listeners que está atenta à ocorrência de eventos
- Assim, um programa deste tipo é composto por:
  - Código que apresenta a interface ao utilizador
  - Os listeners que esperam que ocorram eventos
  - O código que é executado quando ocorre um evento

Interfaces gráficas

- O modelo de programação de interfaces gráficas pode ser representado por:



Interfaces gráficas

- Assim, no geral a programação de interfaces gráficas consiste em:
  - Colocar os componentes desejados
  - Esperar por uma acção do utilizador sobre um dos componentes
  - Quando a acção acontece é detectada pelo event listener que chama o(s) método(s) apropriado(s)
- Esta técnica é geralmente designada por programação dirigida por eventos (event-driven programming)

Interfaces gráficas

- Estas aplicações baseiam-se geralmente em janelas que contêm os diversos componentes.
- A classe JFrame representa janelas em Java:

```
import javax.swing.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
        JFrame jan = new JFrame();
        jan.setSize(dimH,dimV);
        jan.setTitle("Janela");
        jan.setVisible(true);
    }
}
```


Interfaces gráficas

- Para além do setSize() e do setTitle(), a classe JFrame inclui outros métodos que permitem definir as características da janela. Por exemplo, setCursor() ou setBackground().
- A classe JFrame por si só é de pouca utilidade, uma vez que apenas cria a janela sem que esta tenha a capacidade de suportar desenhos ou texto
- Para isso é necessário utilizar um objecto da classe JPanel (ou um seu descendente)

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Desenho extends Jpanel {
    int cx, cy, comp, larg;
    Color cor;
    //Construtor, inicializa as variáveis de instância
    public Desenho(int cx, int cy, int comp, int larg, Color cor)
    {
            this.cx = cx; this.cy = cy; this.comp = comp;
            this.larg = larg; this.cor = cor;
    }
    //Cria o desenho. É chamado sempre que a janela fica visível
    public void paintComponent (Graphics g) {
            super.paintComponent(g);
            g.setColor(cor);
            g.fillRect(cx,cy,comp,larg);
    }
}
```


Interfaces gráficas

- O método paintComponent() desempenha um papel fundamental, uma vez que é chamado automaticamente sempre que é necessário desenhar a janela (por exemplo após ter sido removida uma outra janela que a tapava)
- O método paintComponent() recebe como parâmetro um objecto da classe Graphics, através do qual o intérprete lhe fornece o contexto gráfico em que o desenho vai ser efectuado

Interfaces gráficas

- Para que o desenho do rectângulo possa aparecer na janela, é preciso uma ligação entre esta e o objecto da classe Desenho:

```
import java.awt.*;
import javax.swing.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
      JFrame f = new JFrame();
      f.setSize(dimH,dimV);
      f.setTitle("Janela com rectângulo");
      Desenho d = new Desenho(100,50,100,80,Color.blue);
      Container c = f.getContentPane(); // superfície de desenho
      c.add(d);
      f.setVisible(true);
    }
}
```


Interfaces gráficas

- Suponhamos que se pretendem adicionar à janela três botões, um para tornar a cor do rectângulo mais clara, outro para a tornar mais escura e o terceiro para terminar a aplicação.
- Os botões são definidos pela classe JButton, incluída na biblioteca javax.swing.
- Para adicionar um botão a uma janela, segue-se uma aproximação semelhante à que foi utilizada para adicionar um objecto da classe Desenho, ou seja, utiliza-se o método add() da classe JContainer.

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
            JFrame f = new JFrame();
            f.setSize(dimH,dimV);
            f.setTitle("Janela com rectângulo");
            Desenho d = new Desenho(100,50,100,80,Color.blue);
            Container c = f.getContentPane();
            c.add(d);
            c.add(new JButton("Claro"));
            c.add(new JButton("Escuro"));
            c.add(new JButton("Sair"));
            f.setVisible(true);
    }
}
```


Interfaces gráficas

- Cada um dos componentes (JPanel e JButton) é adicionado à janela sem especificar a sua dimensão ou a sua posição.
- Nesta situação, o intérprete sobrepõe os diversos componentes pela ordem por que são criados, resultando em:

Interfaces gráficas

- Para evitar este comportamento indesejado, os vários componentes gráficos que se pretendem mostrar devem ser colocados dentro de um contentor.
- Este pode ser visto como uma caixa onde vão ser postos todos os elementos de comunicação utilizados na aplicação.
- Após a colocação dos componentes gráficos no contentor, este pode ser adicionado à janela.
- Os contentores são definidos pela classe JPanel, integrada também na biblioteca javax.swing.
- É, então, possível criar uma classe para representar um contentor que contenha os três botões da aplicação:

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Botoes extends JPanel {
    public Botoes()  {
            JButton claro = new JButton("Claro");
            this.add(claro);
            JButton escuro = new JButton("Escuro");
            this.add(escuro);
            JButton sair = new JButton("Sair");
            this.add(sair);
    }
}
```

Interfaces gráficas

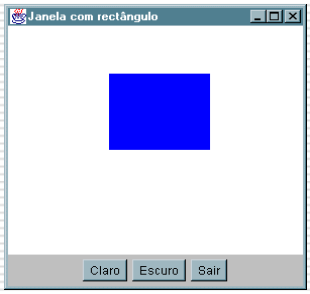
- Uma vez criado o contentor, é necessário adicioná-lo à janela.
- É possível controlar a localização do desenho e do contentor com os botões, de forma a evitar a sua sobreposição.
- Para isso, podem ser utilizados os termos "North", "South", "West", "East" e "Center".
- Estes termos definem o posicionamento relativo entre os vários elementos por analogia com os pontos cardeais e podem ser incluídos como parâmetro no método add().

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
public class Janela {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
            JFrame f = new JFrame();
            f.setSize(dimH,dimV);
            f.setTitle("Janela com rectângulo");
            //Cria e adiciona a área de desenho no centro da janela
            Desenho d = new Desenho(100,50,100,80,Color.blue);
            Container c = f.getContentPane();
            c.add(d, "Center");
            //Cria e adiciona o contentor na parte de baixo da janela
            c.add (new Botoes(), "South");
            f.setVisible(true);
    }
}
```

Interfaces gráficas

- O resultado da execução deste programa é:



Interfaces gráficas

- Os três botões e o desenho aparecem agora correctamente posicionados.
- No entanto, o facto dos botões estarem visíveis não significa que funcionem como esperado.
- A classe JButton define apenas os atributos gráficos dos botões, mas não lhes confere qualquer funcionalidade.
- Para associar um botão a uma acção, é necessário fazer a gestão dos eventos correspondentes

Interfaces gráficas

- A linguagem Java faz a gestão dos eventos provocados pelo utilizador através de objectos destinados à sua recepção e processamento.
- Simplificando, pode-se afirmar que sempre que o utilizador selecciona um componente, este gera um evento que é enviado para um processador de eventos que lhe deve estar associado.
- O processador deve responder à acção do utilizador através de um método denominado actionPerformed().
- No caso da aplicação que tem vindo a ser apresentada, é necessário associar um processador de eventos a cada um dos três botões, o que deve ser feito no construtor da classe Botoes

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Botoes extends Jpanel {
    public Botoes(Desenho d) {
            JButton claro = new JButton("Claro");
            claro.addActionListener(new GereEventos(1, d));
            this.add(claro);
            JButton escuro = new JButton("Escuro");
            escuro.addActionListener(new GereEventos(2, d));
            this.add(escuro);
            JButton sair = new JButton("Sair");
            sair.addActionListener(new GereEventos(3, d));
            this.add(sair);
    }
}
```

Interfaces gráficas

- É importante realçar algumas diferenças entre esta versão da classe Botoes e a anterior:
  - A inclusão do import java.awt.event.*; que indica ao compilador a biblioteca onde estão as classes necessárias para o processamento dos eventos
  - O parâmetro da classe Desenho no construtor da classe Botoes. Este é necessário, porque os botões vão ter que fornecer o endereço da área de desenho ao processador de eventos, para que este possa provocar as correspondentes alterações no desenho
  - A chamada do método addActionListener(), definido na classe JButton, para associar o processador de eventos a cada botão. Este método recebe um objecto da classe GereEventos (o processador de eventos)
  - O construtor da classe GereEventos recebe dois parâmetros, um int que identifica qual dos botões foi accionado e o endereço da área de desenho

Interfaces gráficas

- O código da classe GereEventos é apresentado em seguida:

```
import java.awt.event.*;
public class GereEventos implements ActionListener {
    private int bot;
    private Desenho des;
    public GereEventos(int n, Desenho d)  {
            bot = n;
            des = d;
    }
    public void actionPerformed(ActionEvent e) {
        switch (bot) {
            case 1:des.claro(); break;
            case 2:des.escuro(); break;
            case 3:System.exit(0); break;
        }
    }
}
```

Interfaces gráficas

- De referir que a classe GereEventos implementa uma interface ActionListener
- A interface ActionListener define apenas um método, actionPerformed(), pelo que só este terá que ser implementado em GereEventos

Interfaces gráficas

- O método actionPerformed() é executado sempre que o utilizador seleccionar com o rato um dos botões.
- O primeiro dos parâmetros que recebe permite-lhe identificar qual dos botões foi activado e reagir em conformidade
- O segundo parâmetro é o endereço da área de desenho que o utilizador poderá modificar através da utilização dos botões. Este endereço é obtido no método main(), e deve ser passado ao contentor que contém os botões através do respectivo construtor, que poderá depois passá-lo ao gestor de eventos (da classe GereEventos)
- Obtém-se, assim, uma nova versão para a classe Janela:

Interfaces gráficas

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
public class Janela  {
    private final static int dimH = 300;
    private final static int dimV = 300;
    public static void main(String[] arg) {
            JFrame f = new JFrame();
            f.setSize(dimH,dimV);
            f.setTitle("Janela com rectângulo");
            Desenho d = new
    Desenho(100,50,100,80,Color.blue);
            Container c = f.getContentPane();
            c.add(d, "Center");
            c.add(new Botoes(d), "South");
            f.setVisible(true);
    }
}
```


Interfaces gráficas

- A classe Desenho teve também que sofrer alterações, de modo a incluir os métodos claro() e escuro():

```
public void claro()  {
        cor = cor.brighter();
        repaint();
}
public void escuro() {
        cor = cor.darker();
        repaint();
}
```

Interfaces gráficas

- Exemplo: Applet que replica o texto introduzido num text field

Interfaces gráficas

```
import javax.swing.*;
import java.awt.event.*;
// Classe Mimic demonstra um componente simples e um evento
public class Mimic extends JApplet {
   MimicGUI gui = new MimicGUI (this);
   public void init() {
        gui.init();
   }
// Repete o texto introduzido no text field
   public void update_label() {
     gui.update_label (gui.get_quote());
   }
}
```

Interfaces gráficas

```
// Classe MimicGUI representa a GUI para a applet Mimic

class MimicGUI {

   private JTextField quote = new JTextField(20);
   private JLabel label = new JLabel ("Só sei que nada sei");
   private Mimic applet;
   private Mimic_Action_Listener listener;

// construtor MimicGUI
public MimicGUI (Mimic mimic_applet) {
      applet = mimic_applet;
      listener = new Mimic_Action_Listener (applet);
   }
```


Interfaces gráficas

```
// Cria a GUI e o listener para o text field
public void init() {
      Container c = applet.getContentPane();
      c.add (quote, "North");
      c.add (label, "Center");
      applet.resize (250,100);
      quote.addActionListener (listener);
   }
// Actualiza o texto da label.
public void update_label (String message) {
      label.setText (message);
   }
// Devolve o texto do text field.
public String get_quote() {
      return quote.getText();
   }
}
```

Interfaces gráficas

```
// Classe Mimic_Action_Listener trata dos eventos para a Mimic
class Mimic_Action_Listener implements ActionListener {
   private Mimic applet;

// Prepara o listener guardando uma referência para a applet
// constructor Mimic_Action_Listener
public Mimic_Action_Listener (Mimic listening_applet) {
      applet = listening_applet;
   }

// Actualiza a label quando ocorre um evento
public void actionPerformed (ActionEvent event) {
      applet.update_label();
   }
}
```


Interfaces gráficas

- Há uma interface listener definida para cada tipo de evento, contendo cada uma delas os métodos necessários para responder a esse tipo de evento
- Uma classe listener implementa uma dada interface listener, pelo que deve implementar todos os métodos nela definidos
- Os listeners são adicionados aos componentes
- Quando um componente gera um evento, o método correspondente a esse evento é executado no seu listener
- Um componente pode ter vários listeners (para vários eventos)
- Uma classe listener pode implementar vários interfaces listener, ou seja pode detectar vários tipos de eventos

Interfaces gráficas

- Existem diversos tipos de eventos e cada um deles tem o seu tipo de listener que terá que ser implementado
- Por exemplo:

Acção do utilizador	Evento gerado	Event listener
Seleccionar botão	ActionEvent	ActionListener
Mover o rato	MouseEvent	MouseListener
Rato entra no componente	FocusEvent	FocusListener
Escrever no teclado	KeyEvent	KeyListener
Escrever num TextField	ActionEvent	ActionListener

Interfaces gráficas

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
// Classe Events demonstra a ocorrência de eventos. Espera
    vários eventos
// e escreve mensagens à medida que ocorrem
public class Events extends Applet {
  private TextArea log = new TextArea (15, 100);
  private int count = 0;
  private Label count_label = new Label("", Label.CENTER);
  private Universal_Listener listener = new
    Universal_Listener (log, count_label);
```

Interfaces gráficas

```
public void init() {
      add ("North", new Label ("Event Logging", Label.CENTER));
      add ("Center", log);
      add ("South", count_label);
      addComponentListener (listener);
      log.addFocusListener (listener);
      log.addKeyListener (listener);
      log.addMouseListener (listener);
      log.addMouseMotionListener (listener);
      setSize (800, 400);
    }
} // class Events
```


Interfaces gráficas

```
//  Classe Universal_Listener implementa muitos event interfaces,
class Universal_Listener implements ComponentListener,
      MouseMotionListener, MouseListener, KeyListener,
      FocusListener {
  private TextArea log;
  private Label  count_label;
  private String count_text = "Number of Events: ";
  private int count = 0;

public Universal_Listener (TextArea log, Label count_label) {
      this.log = log;
      this.count_label = count_label;
  } // constructor Universal_Listener
```

Interfaces gráficas

```
private void log_event (AWTEvent event) {
  count++;
  count_label.setText (count_text + count);
  log.append (event.toString() + "\n");
}
public void componentMoved (ComponentEvent event) {
  log_event (event);
}
public void componentHidden (ComponentEvent event) {
  log_event (event);
}
public void componentResized (ComponentEvent event) {
  log_event (event);
}
public void componentShown (ComponentEvent event) {
  log_event (event);
}
```


Interfaces gráficas

```
public void mouseDragged (MouseEvent event) {
  log_event (event);
}
public void mouseMoved (MouseEvent event) {
  log_event (event);
}
public void mousePressed (MouseEvent event) {
  log_event (event);
}
public void mouseReleased (MouseEvent event) {
  log_event (event);
}
public void mouseEntered (MouseEvent event) {
  log_event (event);
}
public void mouseExited (MouseEvent event) {
  log_event (event);
}
public void mouseClicked (MouseEvent event) {
  log_event (event);
}
```

Interfaces gráficas

```
    public void keyPressed (KeyEvent event) {
      log_event (event);
    }
    public void keyReleased (KeyEvent event) {
      log_event (event);
    }
    public void keyTyped (KeyEvent event) {
      log_event (event);
    }
    public void focusGained (FocusEvent event) {
      log_event (event);
    }
    public void focusLost (FocusEvent event) {
      log_event (event);
    }
  }  // class Universal_Listener
```


Interfaces gráficas

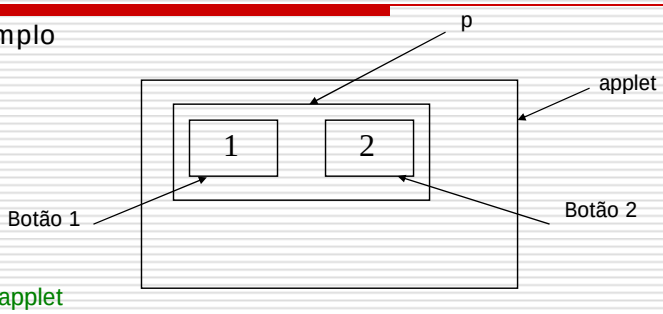
- Um contentor (container) é um tipo especial de componente que serve para agrupar outros componentes (eventualmente outro contentor)
- Por exemplo, uma applet é um contentor e, por isso, podem-lhe ser adicionados outros componentes
- A cada contentor é adicionado um gestor de posicionamento (layout manager) que controla a forma como os seus componentes são mostrados
- Cada contentor tem dois métodos:
  - add (...) - Adiciona um componente ao contentor
  - setLayout (...) - Indica qual o gestor de posicionamento a usar

Interfaces gráficas

- Alguns contentores (JPanel e JApplet) têm que ser ligados a uma superfície gráfica
- Uma applet é ligada a um browser ou a um appletviewer
- Outros contentores (JWindow, JFrame, JDialog) podem ser movidos independentemente
- A utilização de contentores implica que os componentes sejam primeiro adicionados ao contentor, sendo este depois adicionado à applet
- Assim, uma interface gráfica é conseguida pela junção de contentores e outros componentes
- O desenho da interface gráfica é uma parte fundamental do desenvolvimento de aplicações
- Não esquecer que para um utilizador comum a interface é o sistema

Interfaces gráficas

- Exemplo




```
// Na applet
JButton botao1 = new JButton ("1");
JButton botao2 = new JButton ("2");
JPanel p = new JPanel();
p.add (botao1);
p.add (botao2);
getContentPane().add (p);
```

Interfaces gráficas

- Há diversos tipos de componentes que permitem interacção com o utilizador:
  - Etiquetas - Classe JLabel
  - Campos de texto - Classe JTextField
  - Áreas de texto - Classe JTextArea
  - Listas - Class JList
  - Botões - Classes JButton, JCheckbox
  - Scrollbars - Classe JScrollbar
  - Etc.

Interfaces gráficas

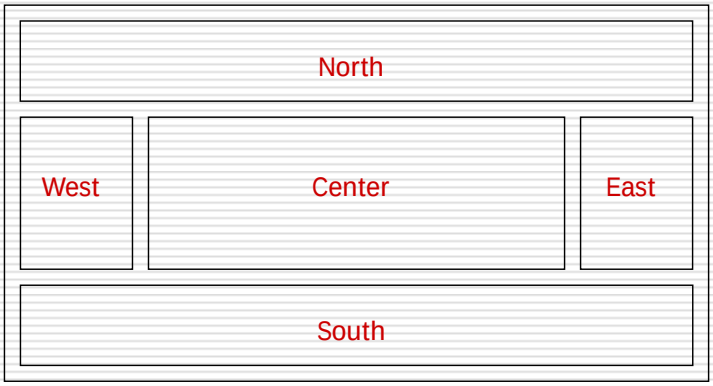
- Um gestor de posicionamento é um objecto que decide como distribuir os componentes num contentor
- Cada contentor tem um gestor de posicionamento por defeito
- Para alterar pode ser utilizado o método setLayout (...)
- Há vários gestores de posicionamento predefinidos:
  - FlowLayout
  - BorderLayout
  - CardLayout
  - GridLayout
  - GridBagLayout

Interfaces gráficas

- FlowLayout
  - Default para JPanel
  - Os componentes são colocados da esquerda para a direita, passando para a linha seguinte quando já não couber na linha corrente
  - Os componentes são centrados
- BorderLayout
  - Default para JApplet
  - Tem cinco locais para colocar componentes: North, South, West, East e Center
  - O programador especifica em qual das áreas quer cada componente
  - O tamanho relativo das áreas é definido pelo tamanho dos componentes que lá são colocados

Interfaces gráficas

- BorderLayout (cont.)



Interfaces gráficas

- GridLayout
  - Os componentes são colocados numa grelha com um determinado número de linhas e colunas
  - Cada componente é colocado numa célula
  - Todas as células têm a mesma dimensão
- CardLayout
  - Os componentes são colocados uns em cima dos outros
  - Apenas um componente está visível em cada momento
  - Os métodos podem controlar qual dos componentes é colocado visível
- GridBagLayout
  - Uma grelha bidimensional de linhas e colunas
  - Nem todas as células são do mesmo tamanho
  - Os componentes podem ocupar várias linhas ou colunas
  - Cada componente é associado com um conjunto de restrições

Interfaces gráficas

- Em resumo, para construir a interface gráfica:
  - Declarar os componentes e os seus listeners
  - Instanciar e inicializar componentes e listeners
  - Adicionar os componentes aos seus contentores (se houver)
  - Adicionar os componentes (e/ou contentores) à applet
  - Registar os listeners nos respectivos componentes
- Para tratar eventos:
  - Escrever métodos na applet para cada tipo de acção que o utilizador possa efectuar
  - Escrever uma classe (ou classes) para cada tipo de listener, incluindo os métodos respectivos

Interfaces gráficas

- Converter applets em aplicações:
  - Mudar o nome do método init para o nome da classe, tornando-o assim no seu construtor (eliminar void do cabeçalho)
  - Alterar o cabeçalho da classe (herda JFrame em vez de JApplet)
  - Criar um método main como:

```
public static void main (String[] args){
   JFrame f = new NomeClasse ();
   f.resize(300,300);
   f.setVisible(true);
  }
```

  - Eliminar o import da classe Applet
  - Adicionar um método handleEvent para tratar o fim da aplicação:

```
public boolean handleEvent (Event e) {
   if (e.id == Event.WINDOW_DESTROY)
         System.exit(0);
   return super.handleEvent(e);
  }
```


Interfaces gráficas

- Converter applets em aplicações (cont.)
  - Adicionar a invocação de um método para definir o gestor de posicionamento a usar. Tipicamente no construtor:

```
setLayout(new FlowLayout());
```

- Numa aplicação podem ser usados menus (ao contrário do que acontece nas applets)