Aula 14: Os pacotes java.io e javax.swing

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

TECNOLOGIAS DE PROGRAMAÇÃO - SETOR DE INFORMÁTICA

Os pacotes java.io e javax.swing

- Ao término desta aula, você será capaz de:
  - ler e escrever bytes, caracteres e Strings de/para a entrada e saída padrão;
  - ler e escrever bytes, caracteres e Strings de/para arquivos;
  - utilizar buffers para agilizar a leitura e escrita através de fluxos;
  - usar Scanner e PrintStream.
  - usar JOptionPane.

Conhecendo uma API

- Vamos passar a conhecer APIs do Java java.io e java.util!
- A parte de controle de entrada e saída de dados (conhecido como io) é orientada a objetos e usa os principais conceitos mostrados até agora: interfaces, classes abstratas e polimorfismo.
  - Polimorfismo: utilizar fluxos de entrada (InputStream) e de saída (OutputStream) para toda e qualquer operação, seja ela relativa a um arquivo, a um campo blob do banco de dados, a uma conexão remota via sockets, ou até mesmo às entrada e saída padrão de um programa.
  - As classes abstratas InputStream e OutputStream definem o comportamento padrão dos fluxos em Java.

InputStream, InputStreamReader e BufferedReader

- Para ler um byte de um arquivo, vamos usar o leitor de arquivo, o FileInputStream.
- Para um FileInputStream conseguir ler um byte, ele precisa saber de onde ele deverá ler.
- A classe obriga você a passar o nome do arquivo pelo construtor: sem isso o objeto não pode ser construído.
  - Por isso temos sempre que colocar *throws IOException*.

```
1  class TestaEntrada {
2     public static void main(String[] args) throws IOException
          {
3       InputStream is = new FileInputStream("arquivo.txt");
4       int b = is.read();
5     }
6   }
```

InputStream, InputStreamReader e BufferedReader

- Para recuperar um caractere, precisamos traduzir os bytes com o encoding dado para o respectivo código unicode, isso pode usar um ou mais bytes.
- Escrever esse decodificador é muito complicado, quem faz isso por você é a classe InputStreamReader.

```
1  class TestaEntrada {
2     public static void main(String[] args) throws IOException
          {
3       InputStream is = new FileInputStream("arquivo.txt");
4       InputStreamReader isr = new InputStreamReader(is);
5       int c = isr.read();
6     }
7   }
```

InputStream, InputStreamReader e BufferedReader

- InputStream tem diversas outras filhas, como ObjectInputStream, AudioInputStream, ByteArrayInputStream, entre outras.
- InputStreamReader é filha da classe abstrata Reader, que possui diversas outras filhas - são classes que manipulam chars.
- Apesar da classe abstrata Reader já ajudar no trabalho de manipulação de caracteres, ainda seria difícil pegar uma String.
- A classe BufferedReader é um Reader que recebe outro Reader pelo construtor e concatena os diversos chars para formar uma String através do método readLine.

InputStream, InputStreamReader e BufferedReader

```
1  class TestaEntrada {
2     public static void main(String[] args) throws IOException
          {
3       InputStream is = new FileInputStream("arquivo.txt");
4       InputStreamReader isr = new InputStreamReader(is);
5       BufferedReader br = new BufferedReader(isr);
6       String s = br.readLine();
7     }
8   }
```

- Aqui lemos apenas a primeira linha do arquivo.
- O método readLine devolve a linha que foi lida e muda o cursor para a próxima linha. Caso ele chegue ao fim do Reader (no nosso caso, fim do arquivo), ele vai devolver null.

InputStream, InputStreamReader e BufferedReader

```
1  class TestaEntrada {
2     public static void main(String[] args) throws IOException
          {
3       InputStream is = new FileInputStream("arquivo.txt");
4       InputStreamReader isr = new InputStreamReader(is);
5       BufferedReader br = new BufferedReader(isr);
6
7       String s = br.readLine(); // primeira linha
8
9       while (s != null) {
10        System.out.println(s);
11        s = br.readLine();
12      }
13
14      br.close();
15    }
16  }
```

Lendo Strings do teclado

- Trocando o InputStream por System.in passamos a ler do teclado!

```
1  class TestaEntrada {
2     public static void main(String[] args) throws IOException
          {
3       InputStream is = System.in;
4       InputStreamReader isr = new InputStreamReader(is);
5       BufferedReader br = new BufferedReader(isr);
6       String s = br.readLine();
7
8       while (s != null) {
9         System.out.println(s);
10        s = br.readLine();
11      }
12    }
13  }
```

A analogia para a escrita: OutputStream

```
1  class TestaSaida {
2     public static void main(String[] args) throws IOException
          {
3       OutputStream os = new FileOutputStream("saida.txt");
4       OutputStreamWriter osw = new OutputStreamWriter(os);
5       BufferedWriter bw = new BufferedWriter(osw);
6
7       bw.write("Testando");
8
9       bw.close();
10    }
11  }
```

- O método write do BufferedWriter não insere o(s) caractere(s) de quebra de linha. Para isso, você pode chamar o método newLine.

Uma maneira mais fácil: Scanner e PrintStream

```
1  Scanner s = new Scanner(System.in);
2  PrintStream ps = new PrintStream("arquivo.txt");
3  while (s.hasNextLine()) {
4    ps.println(s.nextLine());
5  }
```

- A classe Scanner é do pacote java.util!
- Existem duas classes chamadas java.io.FileReader e java.io.FileWriter. Elas são atalhos para a leitura e escrita de arquivos.

Um pouco do javax.swing

- Uma outra forma de entrada de dados é através de interfaces gráficas!
- O Swing traz muitos componentes para usarmos: botões, entradas de texto, tabelas, janelas, abas, scroll, árvores de arquivos e muitos outros.
- A biblioteca do Swing está no pacote javax.swing (inteira, exceto a parte de acessibilidade, que está em javax.accessibility).

JOptionPane

- A classe mais simples do Swing é a JOptionPane que mostra janelinhas de mensagens, confirmação e erros, entre outras.
- Podemos mostrar uma mensagem para o usuário com a seguinte linha:

```
1       JOptionPane.showMessageDialog(null, "Minha mensagem
            !");
```

- A classe JFileChooser é a responsável por mostrar uma janela de escolha de arquivos.

```
1  JFileChooser fileChooser = new JFileChooser();
2  fileChooser.showOpenDialog(null);
```

- O argumento do showOpenDialog indica qual o componente pai da janela de mensagem (pensando em algum frame aberto, por exemplo, que não é nosso caso).
- Esse método retorna um int indicando se o usuário escolheu um arquivo ou cancelou. Se ele tiver escolhido um, podemos obter o File com getSelectedFile:

```
1  JFileChooser fileChooser = new JFileChooser();
2  int retorno = fileChooser.showOpenDialog(null);
3
4  if (retorno == JFileChooser.APPROVE_OPTION) {
5    File file = fileChooser.getSelectedFile();
6    // faz alguma coisa com arquivo
7  } else {
8    // dialogo cancelado
9  }
```

JOptionPane - showMessageDialog

- Exemplos: showMessageDialog

```
1  JOptionPane.showMessageDialog(frame, "Olá, mundo!");
```

```
1  JOptionPane.showMessageDialog(frame, "Olá, mundo!", "
       Aviso", JOptionPane.WARNING_MESSAGE);
```

```
1  JOptionPane.showMessageDialog(frame,  "Olá, mundo!", "
       Erro", JOptionPane.ERROR_MESSAGE);
```

```
1  JOptionPane.showMessageDialog(frame, "Olá, mundo!", "
       Mensagem simples", JOptionPane.PLAIN_MESSAGE);
```

JOptionPane - showOptionDialog

- Exemplos: showOptionDialog

```
1  Object[] options = {"42",
2                      "Não sei",
3                      "Boa pergunta"};
4  int n = JOptionPane.showOptionDialog(frame,
5      "Qual o sentido da vida, o universo e tudo mais?",
6      "Uma pergunta qualquer",
7      JOptionPane.YES_NO_CANCEL_OPTION,
8      JOptionPane.QUESTION_MESSAGE,
9      null,
10     options,
11     options[2]);
```

JOptionPane - showInputDialog

- Exemplos: showInputDialog

```
1  Object[] possibilities = {"feijão", "fritas", "bife"};
2  String s = (String)JOptionPane.showInputDialog(
3                      frame,
4                      "Complete a frase\n"
5                      + "\"Arroz combina com \"",
6                      "Customized Dialog",
7                      JOptionPane.PLAIN_MESSAGE,
8                      icon,
9                      possibilities,
10                     " ");
11
12 //Se o retorno foi uma string
13 if ((s != null) && (s.length() > 0)) {
14     setLabel("Arroz combina com " + s + "!");
15     return;
16 }
17
18 //Se o retorno foi nulo...
19 setLabel("Anda, termine a frase!");
```

- Colocando possibilities como null o resultado é uma caixa de edição! Teste!

Exercícios

- 1. Faça um programa que leia de um arquivo "notas.txt"o nome e a nota de n alunos. O nome do arquivo deve ser escolhido pelo usuário via interface gráfica.
  Crie uma classe Estatística que calcule a média, a variância e o desvio padrão das notas da turma.
  Mostre via interface gráfica as informações calculadas.

Exercícios

- 2. Acrescente uma classe Relatorio. Essa classe é responsável por gerar o arquivo "diferenca_aluno.txt"que contém o nome de cada aluno, com sua nota e a diferença entre sua nota e a média da turma.
- 3. Altere o programa para ler 3 notas de cada aluno.
- 4. Altere a classe Relatorio e acrescente um método para gerar o arquivo "notas_finais.txt"que contém o nome de cada aluno, sua média final e se ele foi aprovado ou reprovado. O aluno é aprovado se sua média é maior ou igual a 60.
- 5. Faça todos os tratamentos de erro necessários!

Exercícios

- 6. Faça um método na classe relatório que procure um aluno no arquivo "notas_finais.txt"e retorne sua média e situação (aprovado/reprovado). O usuário deve inserir o nome do aluno via interface gráfica.
- Observação: Para facilitar os testes, faça um pequeno menu com as funcionalidades do sistema.

Eventos em Java

- Java utiliza um mecanismo especial para tratar eventos, em especial os causados por interações entre usuários e a GUI.
- Eventos são objetos de primeira classe em Java, sendo subclasses de java.util.EventObject e java.awt.AWTEvent.
- A idéia central é permitir que a ocorrência de um mesmo evento possa causar reações distintas e independentes sobre diversos objetos de uma aplicação, de forma flexível, facilitando a expansão e modificação dos sistemas.

Eventos em Java

- Objetos da GUI que podem gerar algum tipo de evento (por exemplo, um botão que pode ser clicado) mantêm uma lista particular de objetos **ouvintes**, interessados em serem notificados da ocorrência de seus eventos.
- A lista é alterada por iniciativa dos ouvintes, que enviam uma mensagem padronizada ao potencial gerador do evento, pedindo para ser registrado na sua lista.
- Esses objetos podem a qualquer momento também pedir para serem removidos da lista.

Eventos em Java

- Quando um evento ocorre (por exemplo, o botão é clicado) o botão, no caso, examina a sua lista de ouvintes, e envia a cada um uma mensagem padrão, passando como parâmetro uma instância de uma classe específica de eventos.
- Cada ouvinte deve ter implementado em sua classe um método para tratar essa mensagem padrão. $\rightarrow$ Interface

Um pouco sobre interface gráfica...

- A interface gráfica é formada por uma grande quantidade de componentes que são colocados em containers. Os contêineres mais usados são janelas (instâncias de JFrame) e painéis (JPanel). Componentes podem ser dispostos (colocados) sobre contêineres, e incluem botões, menus, rótulos (labels), e também outros painéis menores.
- Exemplo: Um painel precisa estar dentro de um container. Contêineres são objetos que podem conter componentes. Uma janela (JFrame) contém um container predefinido que se obtém através do método getContentPane(). Todos os componentes que não são menus devem ser colocados nesse container.

Exemplo

- No nosso primeiro exemplo, o construtor registra o próprio painel como ouvinte dos "eventos de clicar" gerados pelos 3 botões.
- Isso é feito através das mensagens addActionListener(this) enviadas para os 3 botões.
- Essa mensagem faz com que o painel seja incluído nas listas de ouvintes dos 3 botões.
- É definida uma classe JanelaBotoes (extensão de JFrame). O construtor de JanelaBotoes adiciona uma instância de PainelBotoes ao seu container.

Exemplo

Classe do painel com os 3 botões, onde o painel será o ouvinte dos 3 botões:

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class PainelBotoes extends JPanel implements
       ActionListener {
6  // variaveis de instancia:
7          private JButton botaoAmarelo;
8          private JButton botaoAzul;
9          private JButton botaoVermelho;
10
11 //...
```

Exemplo

```
1  //...
2
3  // Construtor
4    public PainelBotoes(){
5          botaoAmarelo = new JButton("Amarelo");
6          this.add(botaoAmarelo); //this é opcional.
               Referencia o painel.
7          botaoAmarelo.addActionListener(this);
8
9          botaoAzul = new JButton("Azul");
10         add(botaoAzul);
11         botaoAzul.addActionListener(this);//registrando o
               painel como ouvinte
12
13         botaoVermelho = new JButton("Vermelho");
14         add(botaoVermelho);
15         botaoVermelho.addActionListener(this);
16     }
17 //...
```

Exemplo

```
1   //...
2  // metodo de ouvinte, para tratar os eventos gerados ao
       clicar um botao
3      public void actionPerformed(ActionEvent evt){
4          Object source = evt.getSource();
5  // A variavel color precisa ter um valor inicial
6  // pois os if's poderiam todos falhar, em teoria.
7          Color color = getBackground();
8          if (source == botaoAmarelo) color = Color.yellow;
9          else if (source == botaoAzul) color = Color.blue;
10         else if (source == botaoVermelho) color = Color.red;
11         setBackground(color);
12         repaint();
13     }
14 }
```

Exemplo

A classe que define a janela:

```
1  import javax.swing.*;
2  public class JanelaBotoes extends JFrame {
3    public JanelaBotoes(){
4        setTitle("Teste de Botoes"); //metodo de JFrame
5        setSize(450, 300);
6        setDefaultCloseOperation(EXIT_ON_CLOSE);
7        add(new PainelBotoes());
8        setVisible(true);
9     }
10 }
```

Exemplo

Classe para testar:

```
1  public class TesteBotoes{
2        public static void main(String[] args){
3              JanelaBotoes frame = new JanelaBotoes();
4        }
5  }
```

Melhorando o exemplo

- Há um defeito básico na forma como programamos o ouvinte dos eventos de botão: cada vez que um botão é clicado, um mesmo ouvinte é avisado.
- O método actionPerformed precisa usar um comando if para testar de qual botão partiu o evento. Isso não é bom porque um mesmo método determina o comportamento de vários botões.
- E em um programa OO bem feito, cada método deve fazer apenas uma ação.

Melhorando o exemplo

- Problema: Para retirar um dos botões, ou incluir um botão novo, ou alterar o comportamento de um deles. Teremos que alterar partes desse único método, potencialmente introduzindo erros nos comportamentos dos demais botões, e obrigando a retestar todos os botões.
- Solução: adotada é criar uma **classe interna** para cada ouvinte.
- Uma classe interna é definida dentro de outra classe, e tem acesso a todas as variáveis e métodos privados da classe hospedeira.

Melhorando o exemplo

PainelBotoes

```
1  import java.awt.Color;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import javax.swing.JButton;
5  import javax.swing.JPanel;
6
7  public class PainelBotoes  extends JPanel {
8
9  private static final long serialVersionUID = 1L;
10 // variaveis de instancia:
11 private JButton botaoAmarelo;
12 private JButton botaoAzul;
13 private JButton botaoVermelho;
```

Melhorando o exemplo

PainelBotoes

```
1     // Construtor
2     public PainelBotoes(){
3           botaoAmarelo = new JButton("Amarelo");
4           this.add(botaoAmarelo); //this é opcional.
                Referencia o painel.
5           botaoAmarelo.addActionListener(new OuvinteAmarelo()
                );
6
7           botaoAzul = new JButton("Azul");
8           add(botaoAzul);
9           botaoAzul.addActionListener(new OuvinteAzul()); //
                ouvinte interno
10          botaoAzul.addActionListener(new ImprimeConsole());
                //ouvinte externo
11
12          botaoVermelho = new JButton("Vermelho");
13          add(botaoVermelho);
14          botaoVermelho.addActionListener(new OuvinteVermelho
                ());
15    }
```

Melhorando o exemplo

Classes ouvintes internas

```
1    class OuvinteAmarelo implements ActionListener{
2            public void actionPerformed(ActionEvent evt){
3                    Color color = Color.yellow;
4                    setBackground(color);
5                    repaint();
6            }
7    }
```

Melhorando o exemplo

Classes ouvintes internas

```
1  class OuvinteAzul implements ActionListener{
2            public void actionPerformed(ActionEvent evt){
3                      Color color = Color.blue;
4                      setBackground(color);
5                      repaint();
6            }
7     }
```

Melhorando o exemplo

Classes ouvintes internas

```
1  class OuvinteVermelho implements ActionListener{
2            public void actionPerformed(ActionEvent evt){
3                        Color color = Color.red;
4                        setBackground(color);
5                        repaint();
6            }
7     }
```

Melhorando o exemplo

```
1  import javax.swing.JFrame;
2
3  public class PainelColoridoComBotoes {
4
5        public static void main(String[] args) {
6              PainelColoridoComBotoes gui = new
                   PainelColoridoComBotoes();
7              gui.go();
8        }
9
10       public void go(){
11             JFrame frame = new JFrame();
12             frame.setTitle("Teste de Botoes"); //metodo de
                   JFrame
13             frame.setSize(450, 300);
14             frame.setDefaultCloseOperation(JFrame.
                   EXIT_ON_CLOSE);
15             frame.add(new PainelBotoes());
16             frame.setVisible(true);
17       }
18 }
```

Criando ouvintes

- Formas de criar os ouvintes dos eventos:
  - O próprio painel é também ouvinte dos seus botões → O primeiro exemplo
  - Criando a classe ouvinte como uma classe interna da classe do painel (inner class); → O exemplo melhorado
  - Construindo uma classe separada para ser o ouvinte dos eventos de botão, um ouvinte externo. → Não é muito elegante!

Eventos de Mouse

- Para cada fonte de eventos da interface com o usuário, Java já tem classes e interfaces pré-definidas!
- Existem duas interfaces para eventos de mouse:
  - MouseListener, para eventos gerados com os botões do mouse.
  - MouseMotionListener, para eventos gerados com o movimento do mouse (cursor).
- Elas podem ser encontradas no pacote java.awt.event. (existe ainda a interface MouseInputListener, que é apenas a junção dos métodos das duas interfaces acima.)

Eventos de Mouse

- No nosso primeiro exemplo, as classes pertencem ao pacote MouseSemAdapter onde todos esses eventos são testados.
- A classe MouseSpy é criada para ser ouvinte de eventos de mouse. Para isso, ela implementa as duas interfaces, MouseListener e MouseMotionListener.
- Eventos:
  - botao pressionado (mouse pressed)
  - botao solto (mouse released)
  - mouse entrou na area da janela (mouse entered)
  - mouse saiu da area da janela (mouse exited)
  - mouse foi clicado na area da janela (mouse pressionado e solto no mesmo lugar - mouse clicked).

Exemplo

```
1  import java.awt.event.MouseEvent;
2  import java.awt.event.MouseListener;
3  import java.awt.event.MouseMotionListener;
4
5  class MouseSpy implements MouseListener, MouseMotionListener
       {
6
7     // eventos da interface MouseListener
8     public void mousePressed(MouseEvent event){
9        System.out.println
10       ("Mouse: Foi pressionado o botao no. " + event.
             getButton()
11         + " em x = "+ event.getX() + " y = " + event.getY()
12       );
13    }
```

Exemplo

```
1     public void mouseReleased(MouseEvent event){
2         System.out.println
3         ("Mouse: Foi solto o botao no. " + event.getButton()
4                   + " em x = "+ event.getX() + " y = " + event
                        .getY()
5         );
6     }
7
8     public void mouseClicked(MouseEvent event){
9         System.out.println
10        ("Mouse: Foi clicado o botao no. " + event.getButton()
11                        + " em x = "+ event.getX() + " y = "
                              + event.getY()
12        );
13    }
```

Exemplo

```
1     public void mouseEntered(MouseEvent event){
2         System.out.println("Mouse: cursor entrou na janela em
              x = "
3             + event.getX() + " y = " + event.getY());
4     }
5
6     public void mouseExited(MouseEvent event){
7         System.out.println("Mouse: cursor saiu da janela em x
              = "
8             + event.getX() + " y = " + event.getY());
9     }
```

Exemplo

```
1
2       // metodos da interface MouseMotionListener
3       public void mouseDragged(MouseEvent event){
4           System.out.println("Mouse no. " + event.getButton() +
                " arrastado em "
5               + event.getX() + " y = " + event.getY());
6       }
7
8       public void mouseMoved(MouseEvent event){
9               System.out.println("Mouse moveu em "
10              + event.getX() + " y = " + event.getY());
11      }
12  }
```

Exemplo

A classe para testar:

```
1  import javax.swing.JFrame;
2
3  public class MouseSpyTest
4  {
5        public static void main (String[] args)
6        {
7              JFrame frame = new JFrame("Testando movimentos
                   do Mouse");
8              frame.setDefaultCloseOperation(JFrame.
                   EXIT_ON_CLOSE);
9              MouseSpy ouvinte = new MouseSpy();
10             frame.addMouseListener(ouvinte);
11             frame.addMouseMotionListener(ouvinte);
12             frame.setBounds(399,0,400,300);
13             frame.setVisible(true);
14       }
15 }
```

A classe Adapter

- No exemplo anterior criamos uma classe ouvinte que realmente implementa todos os 5 métodos definidos na interface MouseListener.
- Mas imagine uma situação em que apenas um tipo de evento deve produzir efeitos, e os demais devem ser ignorados.
- Por exemplo, vamos considerar uma variação do programa anterior MouseSpyTest, em que queremos detectar apenas o evento de botão do mouse pressionado.

A classe Adapter

```
1  class MouseSpy implements MouseListener {
2     public void mousePressed(MouseEvent event){
3        System.out.println
4        ("Mouse: Foi pressionado o botao no. " + event.
             getButton()
5          + " em x = "+ event.getX() + " y = " + event.getY())
               ;
6     }
7     public void mouseReleased(MouseEvent event) { }
8     public void mouseClicked(MouseEvent event){ }
9     public void mouseEntered(MouseEvent event){ }
10    public void mouseExited(MouseEvent event){ }
11 }
```

A classe Adapter

- Mesmo não tendo interesse nos demais métodos, Java nos obriga e incluir todos eles, com corpos vazios.
- Isso traz duas desvantagens: código fica maior, e corremos o risco de esquecer algum método da interface.
- Para contornar essa situação, a API do Java já vem com classes chamadas "adapters", que nada mais são do que classes que implementam interfaces com os corpos todos vazios.

A classe Adapter

- Podemos agora criar nosso ouvinte como sendo uma subclasse do adapter para essa interface, redefinindo apenas o método de interesse.

```
1  import javax.swing.JFrame;
2  import java.awt.event.MouseEvent;
3  import java.awt.event.MouseListener;
4  import java.awt.event.MouseAdapter;
5
6  public class MouseSpy extends MouseAdapter{
7          public void mousePressed(MouseEvent event){
8                          System.out.println
9                           ("Mouse: Foi pressionado o botao no. "
                                + event.getButton()
10                            + " em x = "+ event.getX() + " y = "
                                  + event.getY()
11                          );
12         }
13 }
```

A classe Adapter

A classe teste abaixo testa o efeito de MouseSpy:

```
1  import javax.swing.JFrame;
2  public class MouseSpyTestAdapter{
3        public static void main (String[] args){
4              JFrame frame = new JFrame
5                 ("Testando movimentos do Mouse com Adapter
                      para MousePressed");
6              MouseSpy ouvinte = new MouseSpy();
7              frame.setDefaultCloseOperation(JFrame.
                   EXIT_ON_CLOSE);
8              frame.addMouseListener(ouvinte);
9              frame.setBounds(399,0,400,300);
10             frame.setVisible(true);
11       }
12 }
```

Exercícios

- 1. Fazer todos os exemplos de eventos da aula.
- 2. Dado o exercício de bolinhas, troque a forma de mover uma bolinha para movimentação com o mouse.

Na próxima aula...

Exercícios para a Prova e Prova