FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Electrotécnica

Sistemas de Telecomunicações

2012/2013

Trabalho 0:
Demonstração do ambiente Java
Aprendizagem do desenvolvimento de aplicações

Aula 1 – Primeira aplicação (Versão 2)

Mestrado integrado em Engenharia Eletrotécnica e de Computadores

Luis Bernardo
Paulo da Fonseca Pinto

Índice

1. OBJETIVO

Familiarização com a linguagem de programação Java e com o desenvolvimento de aplicações no ambiente de desenvolvimento NetBeans. O trabalho consiste na realização de um pequeno programa desde o início para perceber como fazer uma aplicação nova. Seguidamente, o aluno faz a introdução parcial do código seguindo as instruções do enunciado, aprendendo a utilizar o ambiente e um conjunto de classes da biblioteca da linguagem Java. Para esta segunda parte, é fornecido um projeto com o início do trabalho, que é completado num conjunto de exercícios.

2. AMBIENTE NETBEANS PARA JAVA

O sistema Java inclui vários pacotes para a criação da interface gráfica de aplicações. As duas mais conhecidas são a `java.awt.*` e `javax.swing.*`. É possível programar aplicações diretamente num editor de texto, mas é conveniente a utilização de um ambiente integrado de desenvolvimento de aplicações que integre um editor de interfaces gráficas, editor de texto, compilador e *debugger*. Escolheu-se para esta disciplina o ambiente NetBeans devido às vastas funcionalidades disponibilizadas, à compatibilidade com Linux, MacOS e Windows, e devido ao preço.

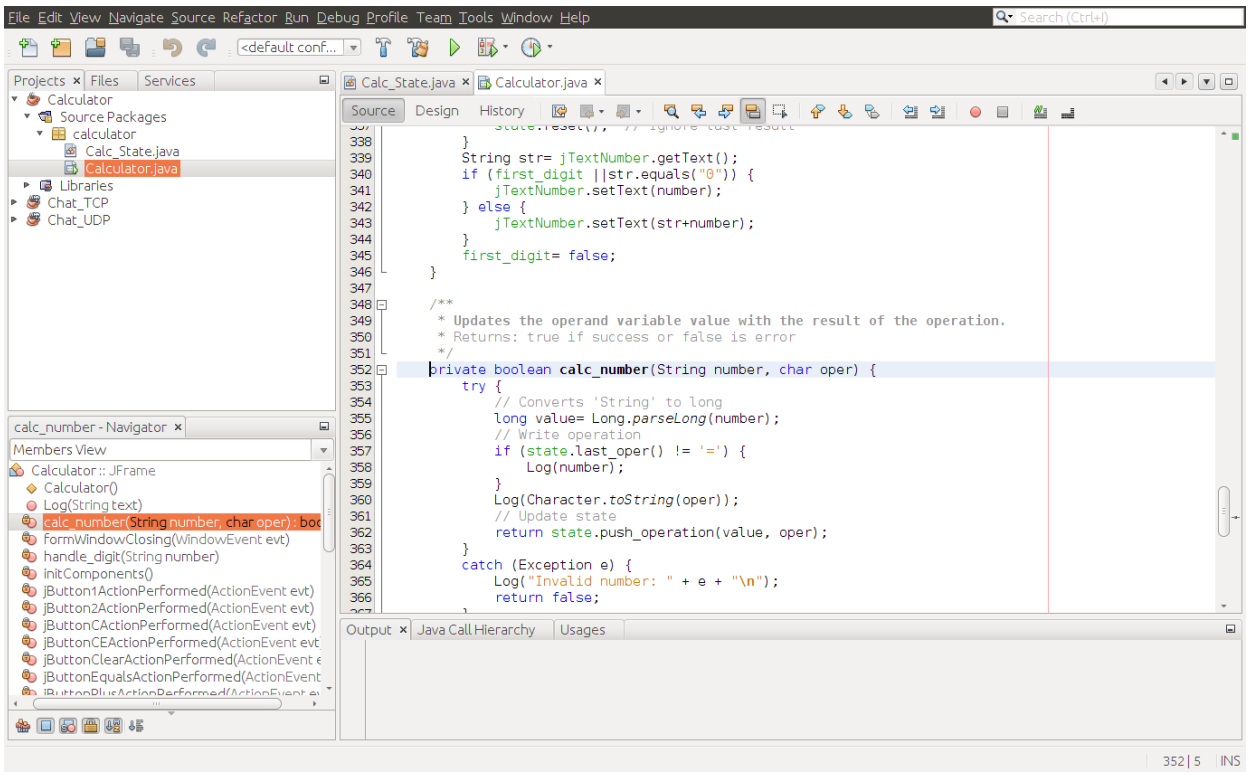
O ambiente NetBeans contém várias janelas que suportam um conjunto variado de opções. Existem quatro modos principais de funcionamento: "*Editing*" (edição de texto), "*GUI Editing*" (edição da interface gráfica), "*Running*" (a correr aplicações); e "*Debugging*" (a correr passo a passo, ilustrada na figura seguinte). Paralelamente existem várias opções realizáveis através de opções do menu, ou de botões. Na imagem está representado o modo de edição, que inclui à esquerda e em cima uma lista de projetos, com a lista de ficheiros de cada projeto. Nos projetos que vão ser realizados na disciplina, existe um ficheiro associado a cada classe. Na caixa "*Projects*" aparece a lista de pacotes (*Source Packages*) por projeto, e para cada pacote, surge a lista de ficheiros: *Calc_State.java* e *Calculator.java*, associados respetivamente às classes com o mesmo nome: `Calc_State` e `Calculator`.

Quando se seleciona um ficheiro surge na janela em baixo (do lado esquerdo) a lista de métodos (funções), constantes e variáveis declaradas nessa classes, permitindo navegar rapidamente para o código da função, representado no editor. A nível do editor, é possível aceder facilmente à ajuda sobre um objeto. Por exemplo, se se introduzisse na função representada, "`dp.`" é aberta uma janela com a lista de métodos suportados na classe DatagramPacket e uma descrição de cada método.

Para compilar  , ou correr com () ou sem () debug, basta usar as teclas indicadas.

A figura seguinte representa a janela de edição de texto do NetBeans, com uma visão hierárquica do código. Marcado a azul aparece código gerado pelo IDE que não pode ser modificado. As palavras reservadas aparecem em letras azuis, os comentários com letras cinzentas, e as strings aparecem com letras vermelhas.

Neste documento vai ser apresentado o desenvolvimento de aplicações de demonstração das funcionalidades do Java, passo a passo, que se recomenda que os alunos sigam no ambiente NetBeans, como forma de aprendizagem. Na parte final, para cada aplicação é proposto um conjunto de exercícios, a ser realizado pelos alunos.



3. O MEU PRIMEIRO PROGRAMA COM JANELAS

Para criar um programa com janelas e botões, comece por criar um projeto novo (chame-o de "*OneButton*"). O projeto é da categoria "Java" e é um projeto "Java Application".

Depois de estar criado, selecione o *package* "*onebutton*" e acrescente um *file Type* (com o botão direito do rato faça "*new*", escolhendo a opção "*other*"). A categoria é "*Swing GUI Forms*" e o *File Type* é "*JFrame Form*". Chame-lhe "JanelaTopo". Foi criado um ficheiro chamado "*JanelaTopo.java*" e as janelas do NetBeans mudaram um pouco:

> Apareceu uma janela central com um quadrado, que é a moldura (*frame*) que se acabou de criar, e duas janelas à direita, sendo a de cima um repositório de outras janelas/classes (do tipo botões, janelas de diálogo, etc.) e a de baixo contém as propriedades do objeto que estiver selecionado na janela central.

Pode ver o código de "*JanelaTopo.java*" escolhendo o botão "*Source*", em vez do "*Design*". Se fizer isso vai ver que este ficheiro também contém a função "*main*" (o ficheiro "*onebutton.java*" também contém). Um modo simples de resolver este problema é eliminar o ficheiro "*onebutton.java*" pura e simplesmente. Elimine-o.

Vamos então começar a colocar objetos gráficos. Começamos por um objeto simples – um rótulo (*label*). Escolha o objeto *label* e leve-o para a moldura. Pode redimensionar o seu tamanho. O rótulo mostra o nome da sua classe. Não é muito interessante, mas haveremos de mudar a mensagem pelo programa propriamente dito. Arraste agora um botão fazendo o mesmo (escolhendo a classe "*Button*", arrastando e dimensionando). Será o nosso botão 1. Desta vez mude o nome dele alterando a propriedade respetiva para "Botao 1". Escolha ainda mais um botão (mude-lhe o nome para "Botao 2") e depois um "*Text Field*". Para o "Text File" alongue o seu comprimento, para ficar grande.

Vamos começar a escrever código.

Se carregar duas vezes no botão 1, é criada uma operação chamada de `jButton1ActionPerformed`. Esta será a operação que será chamada quando o utilizador carregar no "Botao 1". A ação que queremos é que quando se carregar no botão, se escreva a

mensagem “Hello World” no objeto *Text File*. Para isso, dentro desta operação, execute a seguinte instrução

```
jTextField1.setText ("Hello World");
```

Vamos correr o programa.

Quando o tentar correr, o NetBeans vai perceber que “*onebutton.OneButton*” já não existe e pergunta se queremos que a classe principal seja a “*onebutton.JanelaTopo*”. Diga que sim. Quando aparecer a janela, carregue no botão 1 e veja o que acontece.

Termine o programa usando o botão de terminar a janela do seu sistema operativo. O código até ao momento deve estar assim:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package onebutton;

/**
 *
 * @author paulopinto
 */
public class JanelaTopo extends javax.swing.JFrame {

    /**
     * Creates new form JanelaTopo
     */
    public JanelaTopo() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    Generated Code
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        jTextField1.setText ("Hello World");
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        Look and feel setting code (optional)
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new JanelaTopo().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration
}
```

As duas caixas contêm código para dimensionar as várias janelas de acordo com o que foi estabelecido no processo de desenho, e para selecionar a aparência "Nimbus". Dê uma olhadela nesse código.

Já agora, para se mudar o texto do rótulo pode fazer de dois modos: Ou na parte onde os objetos são definidos (na parte ocultada da página anterior com o nome de "*Generated Code*"; ou chamando explicitamente a operação mostrada em baixo noutro sítio conveniente (por exemplo, quando se carrega no botão 1).

```
jLabel1.setText("O meu primeiro programa com janelas");
```

Construa agora o código para o botão 2. Ele deve escrever na "*Text File*" a frase "*Let's rock and roll*". Faça isso, corra o programa e vá carregando num botão e no outro sucessivamente.

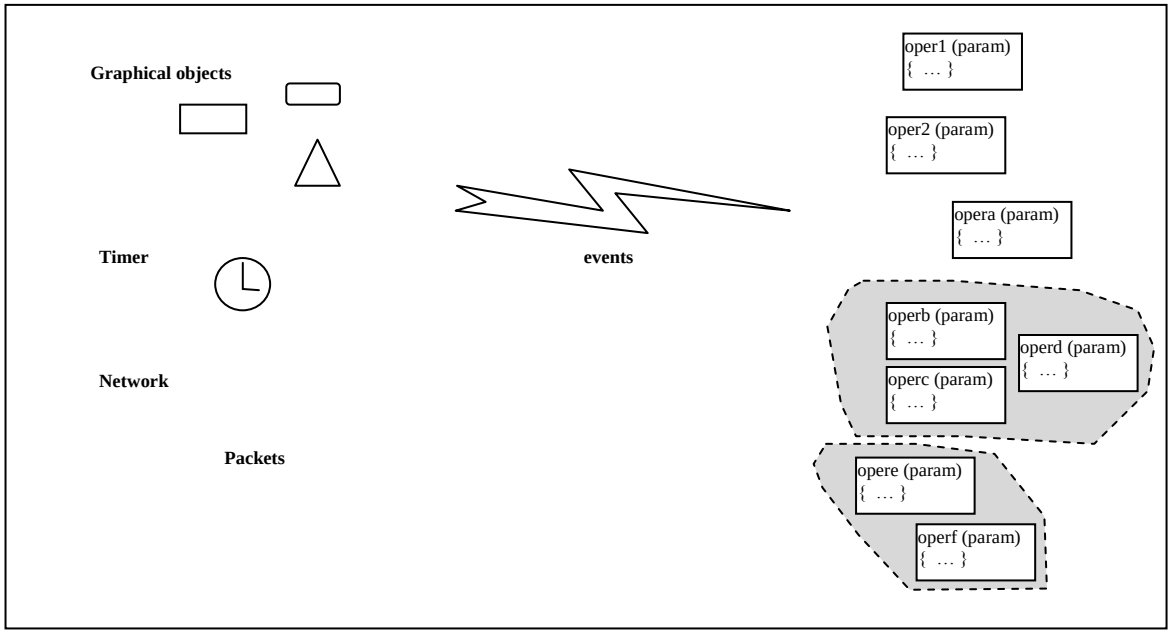
3.1 A ESTRUTURA DO PROGRAMA

Repare que a estrutura do programa é muito diferente do habitual de um programa em C. Agora existem botões, ou outros objectos gráficos, e o código é feito em operações que são chamadas cada vez que se carrega nesses botões.

De um modo geral, os nossos programas vão ser conjuntos de operações. Umas delas estão soltas, outras pertencem a um mesmo objecto. Esses objetos podem ser objectos gráficos, o temporizador, a rede, etc. Esses objetos geram eventos que provocam a chamada

Isso está mostrado na figura em baixo em que se ilustram três operações pertencentes a um objeto, outro objeto com duas operações, e três operações provavelmente pertencentes cada uma a seu objeto (oper1, oper2, opera).

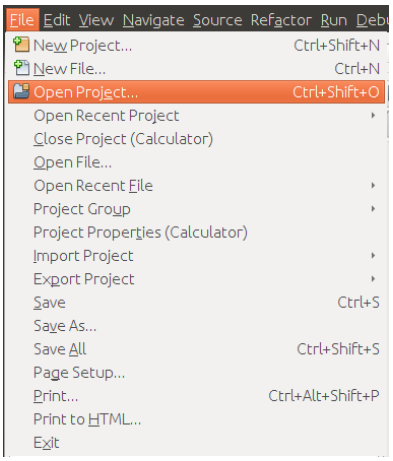
O programador só faz estas operações. Note como é diferente de se programar em C como se aprendeu em Programação de Microprocesadores, em que não se usou a parte gráfica.



4. PRIMEIRA APLICAÇÃO - SOMADORA

Esta aplicação vai ser realizada em duas fases:

1. é realizada apenas uma funcionalidade mínima de entrada de números, soma e registo de operações numa janela e num ficheiro. Esta fase é completamente guiada.
2. é realizada a somadora completa, acrescentando o suporte de mais operações. Esta fase é de exercícios.



É fornecido aos alunos o projeto NetBeans com a primeira fase realizada. Para aceder ao projeto, o aluno deve começar por descompactar o ficheiro *Calculator.zip* para uma diretoria local. De seguida, deve selecionar a opção "`File`"; "`Open Project…`" (conforme está representado à direita), e abrir o projeto `Calculator` a partir da diretoria.

O código do projeto está dentro de um pacote (*package*) com o nome `calculator`, e tem dois ficheiros correspondentes às duas classes que o compõem. A figura no início da página 2 reproduz o que se deverá ver depois de abrir o projeto, e se selecionar o ficheiro *Calculator.java* e o método `calc_number`.

Os alunos deverão analisar esse código seguindo a descrição que é feita nesta secção. Este código servirá de base para realizar a segunda fase, como exercício.

4.1. SOMADORA SIMPLIFICADA

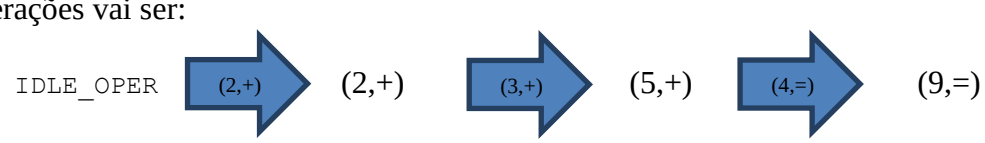
Pretende-se programar uma calculadora que:

- aceite a introdução de números tanto por um conjunto de botões como através de um vetor de caracteres (*string*) diretamente no écran da calculadora,
- suporte um conjunto mínimo de operações ("+" e "="),
- crie um registo de todas as operações numa janela de texto, e opcionalmente, num ficheiro.

A realização vai ser baseada em duas classes: a classe `Calc_State` é usada para manter as parcelas durante o cálculo e a classe `Calculator` define a interface gráfica para o utilizador.

4.1.1. Classe `Calc_State`

A abordagem usada na classe `Calc_State` é dividir as expressões em sequências de pares (*valor* , *operação*) e realizar os cálculos sequencialmente, um par de cada vez, usando a operação `push_operation`. A classe mantém como estado a parcela pendente (`operand`) e a operação pendente (`oper`) (do tipo `char`). No início a operação pendente tem o valor de `IDLE_OPER`, operação nula. Por exemplo, a operação "2+3+4=" é realizada com a composição de três operações: (2, +)(3, +)(4, =). A evolução do estado em `Calc_State` para cada uma das operações vai ser:



O resultado final do cálculo fica guardado no valor de `operand`.

O código da classe `Calc_State` é apresentado na caixa seguinte. Inclui a definição de um construtor (corrido quando se invoca a operação `new Calc_State()`) que inicia o estado do objeto, um método que repõe o estado inicial (`reset()`) e a declaração das duas variáveis da classe privadas (não acessíveis diretamente). Também inclui uma constante

(`Calc_State.IDLE_OPER`), que define um estado sem nenhuma operação pendente na calculadora. Para aceder aos valores das variáveis são definidos dois métodos: `last_oper()` e `operand()`. O método principal é `push_operation(new_operand, new_operation)`, que realiza os cálculos atualizando o estado do objeto, retornando `true` em caso de sucesso, ou `false` caso contrário.

```
public class Calc_State {
    public static final char IDLE_OPER= ' ';    // Constant

    public Calc_State() {     /* Constructor */
        operand= 0;
        oper= IDLE_OPER;
    }

    public char last_oper() {  /* Returns the last operation */
        return oper;
    }


    public long operand() {    /* Returns the current operand */
        return operand;
    }

    public void reset() {     /* Resets the state to the default values */
        operand= 0;
        oper= IDLE_OPER;
    }

    /* Adds a new pair (operand, oper) to the state */
    public boolean push_operation(long new_operand, char new_oper) {
        switch (oper) {    // Calculate the pending operation
            case IDLE_OPER: // First operand
                operand= new_operand;
                break;
            case '+':   // Calculate pending addition
                operand += new_operand;
                break;
            case '=':
                break;
            default:    // Error - operation not supported
                reset();
                return false;
        }
        // Memorizes pending operation
        oper= new_oper;
        return true;
    }

    // Class local variables
    private long operand;
    private char oper;
}
```

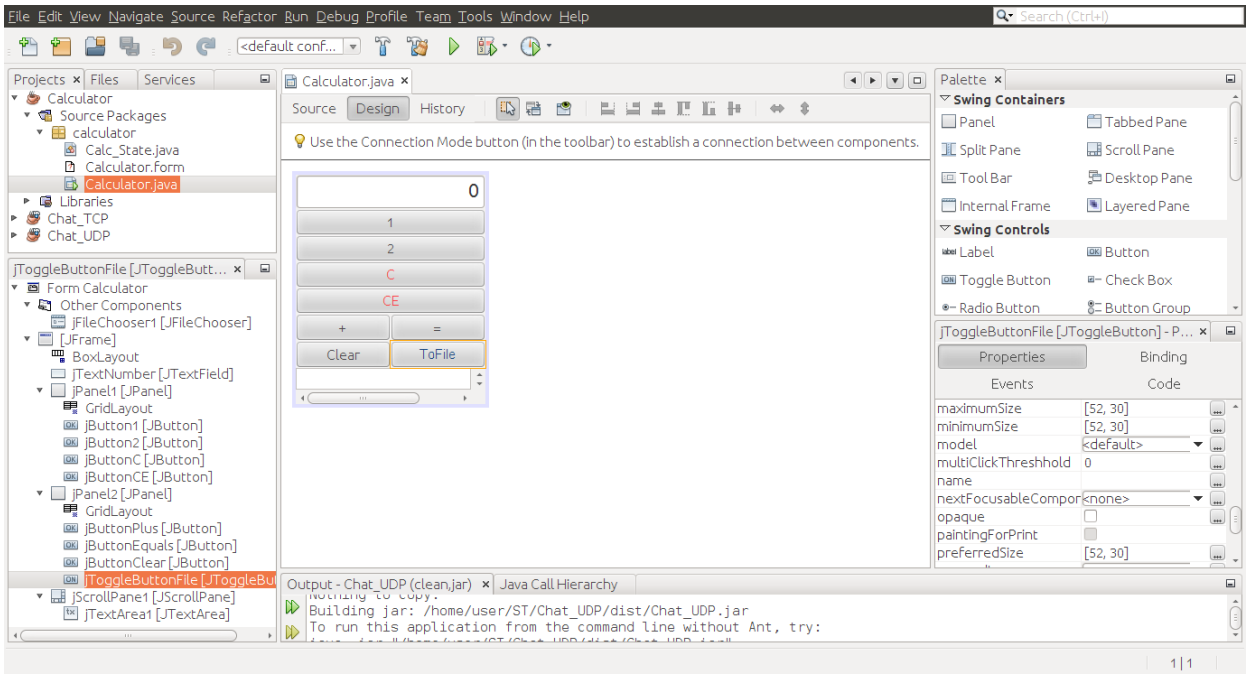
4.1.2. Classe `Calculator` – Interface gráfica

O desenho da interface gráfica é feito usando o modo "*Design*", representado na figura abaixo. Para desenvolver esta aplicação irão ser necessários vários tipos de componentes gráficos (selecionados na caixa "*Pallete*", à direita na imagem) que são adicionados a uma janela do tipo `JFrame`:

- 7 objetos **Button** (da pasta '*Swing Controls*') {para fazer os botões 0-1, +, =, C, etc.}
- 1 objeto **ToggleButton** (da pasta '*Swing Controls*') {para fazer o botão com estado 'ToFile'}
- 1 objeto **Text Field** (da pasta '*Swing Controls*') { Mostrador da calculadora }
- 1 objeto **Text Area** (da pasta '*Swing Controls*') { Janela com o registo das operações }

- 2 objetos **Panel** (da pasta '*Swing Containers*') { Grupos de botões }
- 1 objeto **Scroll Pane** (da pasta '*Swing Containers*') { Barras de deslocamento para *TextArea* }

  Para além disso, foi usado um componente do tipo **File Chooser** (da pasta '*Swing Windows*') para escolher o nome do ficheiro a guardar.



Os objetos gráficos foram dispostos de acordo com a figura acima, tendo-se editado os nomes dos objetos (e.g. o *Text Field* (do tipo `JTextField`) ficou com o nome `jTextNumber`) e o texto dos botões, conforme está representado na janela "*MembersView*" no canto inferior esquerdo da figura acima. Para além disso, foram configuradas as seguintes propriedades:

- objeto `JFrame`
  - ✓ Mudou-se `Layout` para `BoxLayout` com `Axes`= '*Y Axes*'; // Organização vertical dos paineis.
- objetos `jPanel1` e `jPanel2`
  - ✓ Mudou-se `Layout` para `GridLayout`, tendo-se definido o tamanho da grid para (`Columns`=3; `Rows`=4) e (`Columns`=3; `Rows`=2) respetivamente para os painéis 1 e 2.
  - ✓ Mudou-se `MaximumSize`= [210,116] e [210,60] respetivamente para os painéis 1 e 2, para limitar crescimento vertical dos paineis com botões.
  - ✓ Mudou-se `PreferredSize`= [31,116] e [104,60] respetivamente para os painéis 1 e 2.

Modificou-se as fontes e cores de alguns botões. Pode consultar todos os valores das propriedades dos vários objetos no NetBeans.

Por fim, foram criadas as funções de tratamento de eventos de todos os botões, com um clique duplo sobre cada botão, que cria uma função vazia e associa-a ao evento `actionPerformed`. Associou-se também uma função ao evento `windowClosing` do objeto `JFrame`, de maneira a interceptar a terminação da janela.

A classe `Calculator`, é a que contém a função `main`, gerada automaticamente para abrir a janela quando a aplicação arranca.

Para facilitar a interação com a *Text Area* (`jTextArea1`) (a parte inferior onde se escreve mensagens), e possibilitar a escrita em paralelo no terminal e num ficheiro, definiu-se uma função, `public void` **`Log`**`(String str)`, apresentada mais abaixo.

O botão "*Clear*" apaga o conteúdo da *Text Area*:

```
private void jButtonClearActionPerformed(java.awt.event.ActionEvent evt) {
  jTextArea1.setText("");
}
```

4.1.3. Classe `Calculator` – Leitura de Dígitos

Quando se carrega no botão de um dígito, o dígito é memorizado na caixa de texto `jTextNumber`, sendo também usada uma variável Booleana para memorizar se é o primeiro dígito.

```
private boolean first_digit;       // If it is the first digit of a number
```

O método `handle_digit` foi criado para concatenar o dígito `number` à string já existente na caixa de texto cada vez que se prime um botão numérico:

```
private void handle_digit(String number) {
  if ((state.last_oper() == '=') && first_digit) {
    // If the result from the last addition is not used
    Log("");        // Add na empty line to the output
    state.reset();  // Ignore last result
  }
  String str= jTextNumber.getText();    // get the string
  if (first_digit ||str.equals("0")) {  // Ignore leading zeros
    jTextNumber.setText(number);        // Sets the string
  } else {
    jTextNumber.setText(str+number);    // Concatenate digit
  }
  first_digit= false;   // The next digit is not the first!
}
```

O método anterior é usado pelas funções que tratam os botões numéricos. Por exemplo, para o caso do '1' fica:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
  handle_digit("1");
}
```

A função de tratamento do botão '**C**' remove o último dígito do número. O método `jTextNumber.getText()` devolve a string com o conteúdo da caixa de texto, e depois o método `jTextNumber.setText(str)` modifica a string mostrada para o valor de `str`.

```
private void jButtonCActionPerformed(java.awt.event.ActionEvent evt) {
  String str= jTextNumber.getText();    // Get the string
  if (str.length() > 1) {               // Remove the last digit
    str= str.substring(0, str.length()-1);  // substring from 0 to length-1
    jTextNumber.setText(str);           // Set new string
  } else {      // Deleted all digits
    first_digit= true;
    jTextNumber.setText("0");
  }
}
```

4.1.4. Classe `Calculator` – Cálculo

Para realizar o cálculo vai ser usado o objeto `state`, da classe `Calc_State`,

```
private Calc_State state;          // Calculation state
```

inicializado no construtor da classe `Calculator`:

```
state= new Calc_State();           // Allocates a new object Calc_State
```

Foi criado o método `calc_number` para processar a *string* na caixa de texto, validando-a, e para invocar a operação `push_operation` sobre o estado. Adicionalmente, escreve no registo. Observe-se a utilização do `try – catch`: se o número for válido, invoca o método `push_operation` e retorna o que esta retornar; se o número não for válido, corre o código em catch, retornando `false`.

```
private boolean calc_number(String number, char oper) {
  try {
    long value= Long.parseLong(number);        // Converts 'String' to long
    if (state.last_oper() != '=') {
      Log(number); // Write number
    }
    Log(Character.toString(oper));  // Write operation
    return state.push_operation(value, oper);    // Update state
} catch (Exception e) {
    Log("Invalid number: " + e + "\n");
    return false;
  }
}
```

O método que trata o botão '+' tem apenas de invocar o método anterior e preparar a leitura do número seguinte.

```
private void jButtonPlusActionPerformed(java.awt.event.ActionEvent evt)
  calc_number(jTextNumber.getText(), '+');
  first_digit= true;
}
```

O método que trata o botão '=' é semelhante ao anterior, mas adicionalmente, tem de escrever o resultado da operação.

```
private void jButtonEqualsActionPerformed(java.awt.event.ActionEvent evt)
  if (calc_number(jTextNumber.getText(), '=')) {
    Log(Long.toString(state.operand()));   // Writes the result of the operation
    jTextNumber.setText( Long.toString(state.operand()) );   // Update the number
  }
  first_digit= true;   // Wait for first digit of next number
}
```

4.1.5. Classe `Calculator` – Escrita no ficheiro

A escrita no ficheiro vai requerer duas variáveis adicionais: o descritor de ficheiro `f` e o objeto para escrita `os`. O ficheiro está ativo quando `os!=null`; caso contrário não está activo e não se escreve no ficheiro.

```
private File f;                    // File object
private BufferedWriter os;         // File writting objects
```

O método `start_writing_to_file` abre uma janela permitindo ao utilizador escolher o ficheiro onde quer guardar o registo das operações, abrindo de seguida o objeto de escrita no ficheiro `os`. Retorna `true` em caso de sucesso, ou `false` caso o ficheiro não seja aberto.

```
private boolean start_writing_to_file() {
  try {
    // Open FileChooser dialog window na wait for filename selection
    if (jFileChooser1.showSaveDialog(this) == JFileChooser.APPROVE_OPTION) {
      f= jFileChooser1.getSelectedFile();   // Get the filename selected
      Log("Writing to: "+f+"\n");
      os= new BufferedWriter(new OutputStreamWriter(
                        new FileOutputStream(f), "8859_1"));  // Open writing device
      return true;   // Success - writing to file
    }
  } catch (Exception e) {  // Error during file selection or opening
    System.err.println("Error selecting output file: "+e);
  }
  return false;
}
```

O método `stop_writing_to_file` faz a operação inversa: fecha os objetos de escrita e liberta a memória, colocando-os a `null`.

```
private void stop_writing_to_file() {
  try {
    if (os != null) {  // If the writing device is open
      os.close();  // Close file writing device
      os= null;    //     and free memory setting references to null
      f= null;
      Log("Stopped writing\n");
    }
  } catch (Exception e) {
    System.err.println("Error: "+e);
  }
}
```

O controlo da escrita no ficheiro é feito através do botão com estado "*ToFile*". A função seguinte trata os eventos do botão com estado, iniciando ou terminando a escrita para o ficheiro.

```
private void jToggleButtonFileActionPerformed(java.awt.event.ActionEvent evt) {
  if (jToggleButtonFile.isSelected()) {   // If it is selected, start writing
    if (!start_writing_to_file()) {
      jToggleButtonFile.setSelected(false); // If failed, set button off
    }
  } else {        // If it is not selected
    stop_writing_to_file();
  }
}
```

A escrita das operações no ficheiro foi realizada na função `Log`, que escreve simultaneamente para a *TextArea*, para o terminal e para o ficheiro, caso esteja aberto:

```
public void Log(String text) {
  jTextArea1.append(text+"\n");      // Write to the TextArea
  System.out.println(text);          // Write to the terminal

  if (os != null) {                  // If file is open, write to file
    try {
      os.write(text + "\n");
    } catch (IOException ex) {       // Close file, if write failed
      jTextArea1.append("Error writing to file: "+ex+"\n");
      stop_writing_to_file();
    }
  }
}
```

A escrita do conteúdo no ficheiro ocorre apenas quando se fecha o ficheiro. Para garantir que não se perde nada quando se fecha a aplicação, foi acrescentada a função seguinte para tratar o evento `windowClosing` do objeto `JFrame`, gerado quando se fecha a janela.

```
private void formWindowClosing(java.awt.event.WindowEvent evt) {
  // Main window is closing - close file if it is open
  stop_writing_to_file();
}
```

4.1.6. Classe `Calculator` – Arranque e terminação da aplicação

O arranque da aplicação ocorre a partir da função `main`, integralmente criada pelo *NetBeans* durante a criação do projeto. Nesta função, e dentro de uma tarefa gerida pelo ambiente gráfico, é criado um novo objeto do tipo `Calculator` e é tornado visível.

```
public static void main(String args[]) {
  java.awt.EventQueue.invokeLater(new Runnable() { // Create and display the form
            public void run() {
                new Calculator().setVisible(true);
            }
  });
}
```

O valor inicial das variáveis da classe Calculator é definido no construtor:

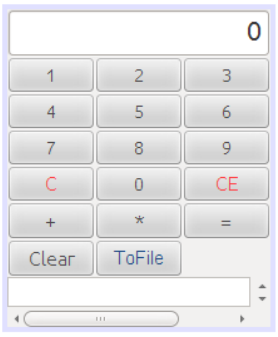
```
public Calculator() {
  initComponents();     // Start graphical window; created by NetBeans
  first_digit= true;          // Waiting for first number
  state= new Calc_State();    // Create object with operand and operation state
}
```

Durante o funcionamento da calculadora, é possível repor o estado inicial (excepto a gravação para ficheiro) através do botão CE:

```
private void jButtonCEActionPerformed(java.awt.event.ActionEvent evt)
  state.reset();
  Log("");                // Add an empty line
  first_digit= true;
  jTextNumber.setText("0");
}
```

4.2. Somadora Completa – Exercício para primeira aula

Na segunda fase desta aula pretende-se completar a calculadora iniciada anteriormente, completando o conjunto de teclas numéricas (‘0’ a ‘9’) e acrescentando uma operação nova (‘*’).



4.2.1. Botões numéricos ‘0’ a ‘9’

Como primeiro exercício, os alunos devem modificar a interface gráfica anterior, de maneira a terem todos os botões numéricos. Para tal, devem abrir o ficheiro “Calculator.java” selecionando o modo “*Design*” do editor. De seguida, devem acrescentar os oito objetos “*Button*” à *form*, no primeiro painel, modificando o seu nome e texto, de acordo com a figura à direita.

Numa segunda fase, devem criar as funções (métodos) de tratamento

dos novos botões, preenchendo o código correspondente. Para resolver este problema, sugere-se uma leitura à secção 4.1.3.

4.2.2. Operação multiplicação

Como segundo exercício, pretende-se que se acrescente a operação multiplicação à calculadora. Este exercício tem três fases distintas:

- Acrescentar botão '*'
- Programar cálculo sem precedência
- Programar cálculo com precedência

A primeira fase é semelhante ao exercício anterior: acrescentar um botão à lista de operações. A segunda e terceira fases obrigam a definir um método para tratar o botão '*' e a reprogramar a classe `Calc_State`, de maneira a esta passar a reconhecer a operação multiplicação. A segunda fase deve ignorar a precedência da multiplicação face à adição e calcular sequencialmente as operações; isto é, "2+2*2" deve dar o resultado 8.

A terceira fase vai novamente incidir sobre a classe `Calc_State`, mas obriga a acrescentar mais variáveis à classe, para memorizar até duas operações pendentes. No caso de "(2,+)(2,*)(2,=)" é necessário guardar (2,+); (2,*) e só depois de receber (2,=) é que se vai realizar os cálculos, obtendo-se o resultado correcto (6).

4.2.3. Temporização

Crie um temporizador usando procedimentos parecidos com os mostrados no documento de introdução ao Java. O objetivo é fazer com que a calculadora execute a operação que estiver pendente caso o utilizador não prima nenhuma tecla durante 10 segundos a meio de um cálculo. É como se o utilizador tivesse premido o botão '='. Não se esqueça de reiniciar o relógio, cada vez que o utilizador carregar em certas teclas e de pará-lo quando carregar noutra tecla.