Desenvolvimento de Sistemas

Manipulação de arquivos: entradas e saídas de dados, leitura e escrita de arquivos

Entradas e saídas de dados

O manuseio de arquivos é parte integrante de qualquer linguagem de programação, pois ele permite armazenar a saída de qualquer programa específico em um arquivo e também realizar diversas operações nele. Em síntese, manipulação de arquivos significa ler e gravar dados em um arquivo.

Em Java, com a ajuda da classe **File**, é possível trabalhar com arquivos. Essa classe de arquivo está dentro do pacote **java.io**. A classe **File** pode ser usada criando um objeto da classe e especificando o nome do arquivo. Confira a seguir um detalhamento dos principais métodos da classe **File**.

Métodos de classe de arquivo File

A tabela a seguir descreve vários métodos de classe de arquivo que permitem realizar várias operações na linguagem Java.

Métodos	Descrição	Retorno
canRead()	Testa se o arquivo é legível ou não.	**Boolean**
canWrite()	Testa se o arquivo é gravável ou não.	**Boolean**
createNewFile()	Cria um arquivo vazio.	**Boolean**
delete()	Exclui um arquivo.	**Boolean**
exists()	Testa se o arquivo existe ou não.	**Boolean**
length()	Retorna o tamanho do arquivo em *bytes*.	**Long**
getName()	Retorna o nome do arquivo.	**String**
list()	Retorna uma matriz dos arquivos no diretório.	**String[]**
mkdir()	Cria um novo diretório.	**Boolean**
getAbsolutePath()	Retorna o caminho absoluto do arquivo.	**String**

Operações de arquivo em Java

Analise agora as várias operações que podem ser executadas em um arquivo em Java:

- Criar um arquivo
- Ler conteúdo de um arquivo
- Gravar em um arquivo
- Excluir um arquivo

Agora você estudará em detalhes cada uma das operações citadas. Para cada um dos exemplos, crie um projeto Java com Ant. Se preferir, crie um projeto apenas e depois modifique o método **main()** com os códigos propostos. Não se esqueça de usar a ferramenta de *import* do NetBeans para as classes incluídas (a maioria das classes com as quais você trabalhará estão no pacote **java.io**).

1. Criar um arquivo

Para criar um arquivo em Java, você pode usar o método **createNewFile()**. Se o arquivo for criado com sucesso, ele retornará um valor booleano *true* e *false* se o arquivo já existir. Veja uma demonstração de como criar um arquivo em Java. Para testar, primeiro crie uma pasta na unidade padrão do sistema (**C:** no Windows) com o nome “teste”. O resultado após a execução deve ser “arquivo.txt” criado nessa pasta.

```
      //importando a classe File
      import java.io.File;

      //Importe a classe IOException para lidar com erros
      import java.io.IOException;

      public class Criar {
          public static void main(String[] args)
          {

              try {
                  File obj = new File("C://teste//arquivo.txt");
                  if (obj.createNewFile()) {
                      System.out.println("Arquivo criado: " + obj.getName
());
                  }
                  else {
                      System.out.println("Arquivo já existe.");
                  }
              }
              catch (IOException e) {
                  System.out.println("Ocorreu um erro." + e.getMessage
());
              }
          }
      }
```

2. Ler conteúdo de um arquivo

Você usará a classe **Scanner** para ler o conteúdo de um arquivo. Veja uma demonstração de como ler o conteúdo de um arquivo em Java. Para testar, experimente abrir em um editor o “arquivo.txt” criado no teste anterior e escrever algumas linhas de texto nele. O código deve mostrar no console do NetBeans o texto presente no arquivo.

```
//importando a classe File
import java.io.File;

//Importe a classe FileNotFoundException para lidar com erros
import java.io.FileNotFoundException;

//Importe a classe Scanner para ler o conteúdo de arquivos de texto
import java.util.Scanner;

public class Ler {
    public static void main(String[] args)
    {
        try {
            File obj = new File("C://teste//arquivo.txt");
            Scanner Reader = new Scanner(obj);
            while (Reader.hasNextLine()) {
                String data = Reader.nextLine();
                System.out.println(data);
            }
            Reader.close();
        }
        catch (FileNotFoundException e) {
            System.out.println("Ocorreu algum erro.");
        }
    }
}
```

3. Gravar em um arquivo

Utiliza-se a classe **FileWriter** junto com seu método **write()** para gravar algum texto no arquivo. Essa classe herda da classe **OutputStream**, que será mais bem detalhada na seção “Leitura e escrita de arquivos”.

Observe a demonstração de como escrever texto em um arquivo em Java. Após a execução do código, crie (caso não exista) na pasta **Teste** um arquivo de texto (“arquivo.txt”) contendo a frase “Gravando dados no arquivo!”.

```
      //importando a classe FileWriter
      import java.io.FileWriter;

      //Importe a classe IOException para lidar com erros
      import java.io.IOException;

      public class Gravar {
          public static void main(String[] args)
          {
              try {
                  FileWriter Writer = new FileWriter("C://teste//arquivo.
txt");
                  Writer.write("Gravando dados no arquivo!");
                  Writer.close();
                  System.out.println("Sucesso ao gravar no arquivo.");
              }
              catch (IOException e) {
                  System.out.println("Ocorreu algum erro.");
              }
          }
      }
```

4. Excluir um arquivo

Utiliza-se o método **delete()** para excluir um arquivo. A seguir, veja uma demonstração de como excluir um arquivo em Java.

```
      //importando a classe File
      import java.io.File;

      public class Excluir {
          public static void main(String[] args)
          {
              File obj = new File("C://arquivo.txt");
              if (obj.delete()) {
                  System.out.println("Excluido o arquivo : " + obj.getNam
e());
              }
              else {
                  System.out.println("Falha ao excluir o arquivo");
              }
          }
      }
```

Crie um programa em Java para manipulação de leitura e escrita de arquivos, usando as classes **File** e **FileWritter**. O programa deve ter em console um menu com as opões:

1. Ler o conteúdo do arquivo
2. Escrever no arquivo
3. Sair

Quando for digitada a opção 1, o programa deve simplesmente escrever no console as informações contidas no arquivo. Caso o usuário digite 2, o programa permitirá que o usuário escreva um texto para salvar no arquivo. Por fim, se digitar 3, a execução encerra. Lembre-se sempre de exibir o menu novamente depois de o usuário selecionar as opções 1 e 2.

Leitura e escrita de arquivos

Todas as linguagens de programação, como o Java, fornecem suporte para E/S (entrada e saída), em que o programa do usuário pode receber entradas de um teclado e então produzir uma saída na tela do computador.

Como visto anteriormente, a manipulação de arquivos com a classe **File** contém diversas funcionalidades, porém em arquivos limitados. Para manipular arquivos mais complexos, você precisa se familiarizar com um conceito conhecido como **Stream**.

- Em Java, uma sequência de dados é conhecida como fluxo.
- Esse conceito é usado para realizar operações de E/S em um arquivo.
- Existem dois tipos de fluxos: entrada e saída.

Conforme descrito anteriormente, um fluxo pode ser definido como uma sequência de dados. O **InputStream** é usado para ler dados de uma fonte e o **OutputStream** é usado para gravar dados em um destino. A seguir, veja uma hierarquia de classes para lidar com fluxo de entrada e saída.

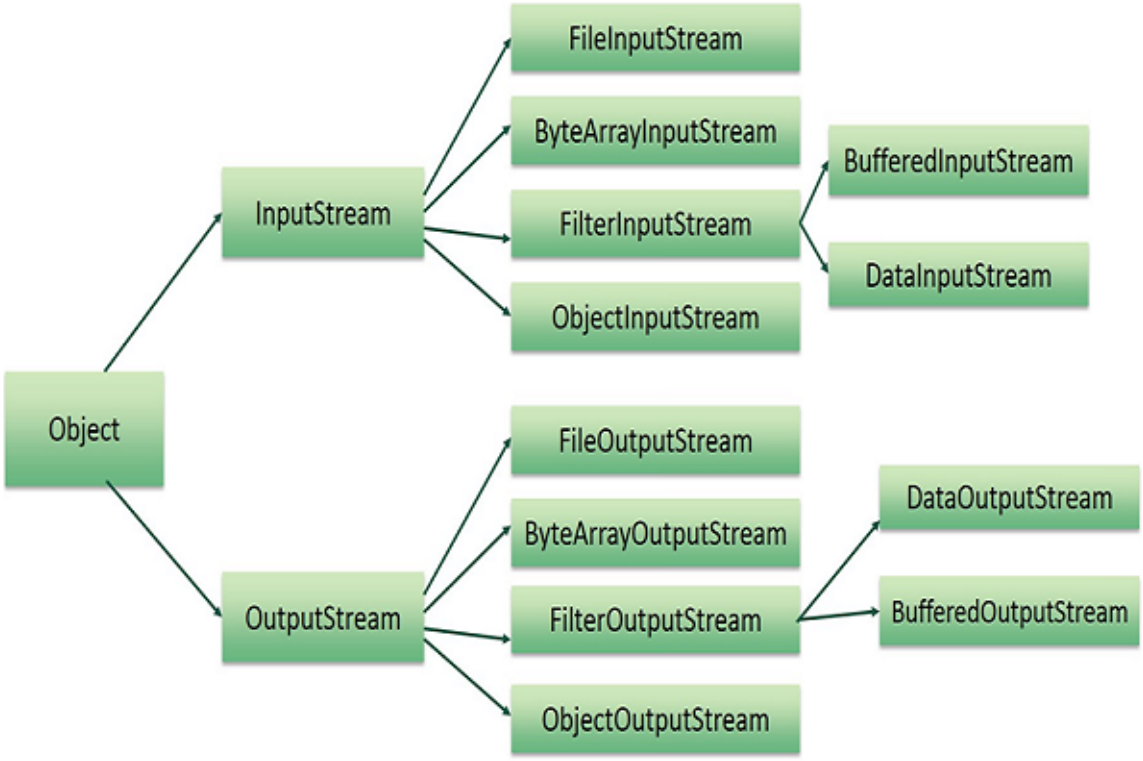

Figura 1 – Classes abstratas **InputStream** e **OutputStream**

Fonte: Pitter (2017)

Diferenciando os fluxos de acordo com o tipo de dados

Com base no tipo de dados, os fluxos podem se comportar de duas formas:

1. Fluxo de *bytes*

Esse fluxo é usado para ler ou gravar dados de *bytes* de 8 *bits* (*binary digit*). O fluxo de *bytes* é novamente subdividido em dois tipos, que são os seguintes:

- Fluxo de entrada de *bytes*: usado para ler dados de *bytes* de diferentes dispositivos.
- Fluxo de saída de *byte*: usado para gravar dados de *byte* em diferentes dispositivos.

Embora existam muitas classes relacionadas com fluxo de *bytes*, as classes mais usadas na linguagem Java são: **FileInputStream** e **FileOutputStream**.

A seguir, veja um exemplo que utiliza essas duas classes para copiar os dados de um arquivo de entrada denominado “input.txt” em um arquivo de saída “output.txt”.

```
//importando a classe para manipulação de arquivo
import java.io.*;

public class Copiar {

  public static void main(String args[]) throws IOException {

      FileInputStream in = null;
      FileOutputStream out = null;

      try {
          in = new FileInputStream("C:\\teste\\input.txt");
          out = new FileOutputStream("C:\\teste\\output.txt");

          int c;
          while ((c = in.read()) != -1) {
              out.write(c);
          }
      } finally {
          if (in != null) {
              in.close();
          }
          if (out != null) {
              out.close();
          }
      }
  }
}
```

Conforme disposto anteriormente, criou-se uma instância de **FileInputStream** passando como argumento a localização e o nome do arquivo que se deseja ler. O método **read()** do **InputStream** retorna um valor inteiro que contém o **byte** correspondente que foi lido. O detalhe é que, enquanto o valor lido for diferente de -1, significa que a leitura ainda não terminou, por isso o armazenamento deve ser feito dentro da instância de **FileOutputStream** por meio do método **write()**.

2. Fluxo de caracteres

Esse fluxo é usado para ler ou gravar dados de caracteres com 16 *bits* Unicode. O fluxo de caracteres é novamente subdividido em dois tipos, que são os seguintes:

- Fluxo de entrada de caracteres: usado para ler dados de caracteres de diferentes dispositivos.
- Fluxo de saída de caracteres: usado para gravar dados de caracteres em diferentes dispositivos.

Embora existam muitas classes relacionadas a fluxos de caracteres, as classes mais usadas são **FileReader** e **FileWriter**. Ainda que internamente o **FileReader** use o **FileInputStream** e o **FileWriter** use o **FileOutputStream**, aqui, a principal diferença é que o **FileReader** lê dois *bytes* por vez e o **FileWriter** grava dois *bytes* por vez.

É possível reescrever assim o exemplo anterior, que faz o uso dessas duas classes para copiar um arquivo de entrada em um arquivo de saída:

```
//importando a classe para manipulação de arquivo
import java.io.*;

public class Copiar2 {

  public static void main(String args[]) throws IOException {
      FileReader in = null;
      FileWriter out = null;

      try {
        in = new FileReader("C:\\teste\\input.txt");
        out = new FileWriter("C:\\teste\\output.txt");

        int c;
        while ((c = in.read()) != -1) {
            out.write(c);
        }
      }finally {
        if (in != null) {
            in.close();
        }
        if (out != null) {
            out.close();
        }
      }
  }
}
```

Conclui-se, portanto, que **FileInputStream** e **FileOutputStream** são adequadas para arquivos em geral e contam com mais recursos; **FileReader** e **FileWriter** são destinadas mais especificamente a arquivos textuais.

Criando um minieditor de texto com Java Swing

Para criar um editor de texto simples em Java Swing, você pode usar os componentes **JFrame**, **JTextArea**, **JMenuBar** e **JMenu**, nos quais são adicionados itens no menu com o componente **JMenuItems**. Todos os itens do

menu terão **actionListener** para detectar qualquer ação.

Haverá uma barra de menus que conterá um menu e um botão:

Passos importantes a serem considerados no momento da implementação:

- Primeiro, crie um **JFrame** intitulado “MiniEditor”; adicione uma área de texto com **JTextArea** e uma barra de menu **JMenuBar** com duas opções: “Arquivo” e “Fechar”.
- A opção “Arquivo” (**JMenu**) contém três itens de menu: “Novo”, “Abrir” e “Salvar” (**JMenuIntem**). Adicione um ouvinte de ação a todos os itens de menu (usando a função **addActionListener())** para detectar qualquer ação.
- Adicione a área de texto ao quadro usando a função **add**; defina o tamanho do quadro para 500 X 500 usando a função **setSize(500,500)** e, em seguida, exiba o quadro usando a função **show**.

Confira como as funções do menu serão invocadas:

- Ao selecionar o item de menu **Abrir**, o seletor de arquivo da caixa de diálogo **Abrir** será aberto. Após selecionar um arquivo, um leitor de arquivo e um leitor de *buffer* lerão o arquivo e definirão o texto da área de texto para o conteúdo do arquivo.
- Ao selecionar o item de menu **Salvar**, o seletor de arquivo da caixa de diálogo **Abrir** será aberto. Após selecionar um arquivo, o gravador de arquivos (gravador em *buffer*) gravaria o conteúdo da área de texto no arquivo e fecharia o gravador de arquivos e o gravador em *buffer*.
- Se o item de menu **Novo** for selecionado, o texto da área de texto ficará em branco. Se o item de menu **Fechar** for selecionado, o quadro é fechado usando a função **setVisible(false)**.

Estes são os passos de implementação no NetBeans:

- Crie um projeto com nome de “MineEditor”, no qual a classe **MineEditor** estenderá a classe **JFrame** implementando **ActionListener**.

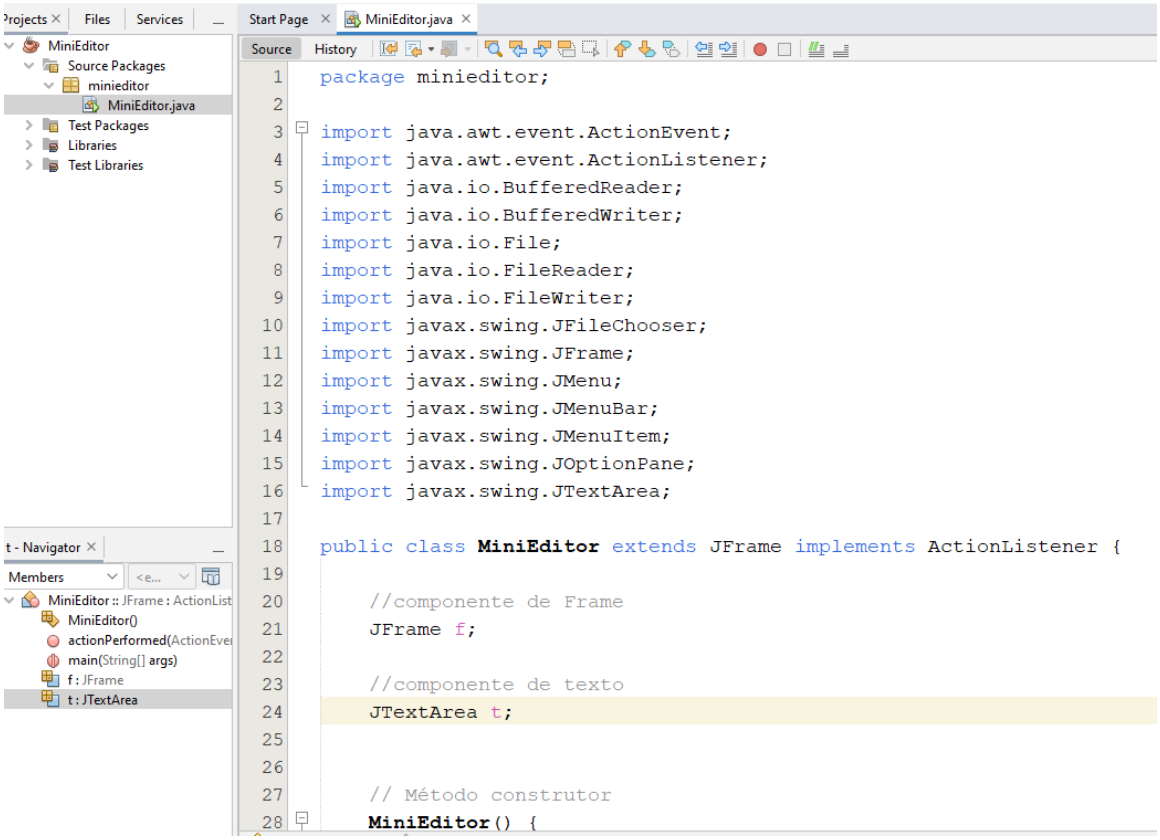

Figura 2 – Criando **MiniEditor** em Java Fonte: NetBeans (2022)

- Realize os ajustes de *imports* e o desenvolvimento do código. Para implementar o **MiniEditor**, não será necessário utilizar componentes visuais apenas via código, conforme código detalhado a seguir.

Classe **MineEditor** com comentário de explicações de classes e métodos utilizados nas funcionalidades desenvolvidas:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JTextArea;

public class MiniEditor extends JFrame implements ActionListener {

  //componente de Frame
  JFrame f;

  //componente de texto
  JTextArea t;

  // Método construtor
  MiniEditor() {

      // Criando um frame
      f = new JFrame("MineEditor");

      // Criando o componente text
      t = new JTextArea();

      // criando a barra de menu
      JMenuBar mb = new JMenuBar();

      // criando o menu arquivo
      JMenu m1 = new JMenu("Arquivo");

      //criando os itens do menu arquivo
      JMenuItem mi1 = new JMenuItem("Novo");
      JMenuItem mi2 = new JMenuItem("Abrir");
      JMenuItem mi3 = new JMenuItem("Salvar");

      // adicionando as actions listener
      mi1.addActionListener(this);
      mi2.addActionListener(this);
      mi3.addActionListener(this);
```

```
            //adicionando as opções no meu arquivo
            m1.add(mi1);
            m1.add(mi2);
            m1.add(mi3);

            //criando o menu fechar
            JMenuItem mc = new JMenuItem("Fechar");

            // adicionando as actions listener
            mc.addActionListener(this);

            //adicionando as opções do menu na barra de menus
            mb.add(m1);
            mb.add(mc);

            //especificando as configurações do frame, com barra de menus e t
amanho
            f.setJMenuBar(mb);
            f.add(t);
            f.setSize(500, 500);
            f.show();
        }

        // quando o botão for pressionado no menu
        public void actionPerformed(ActionEvent e) {
            String s = e.getActionCommand();

            switch (s) {

                case "Novo":
                    t.setText("");
                    break;

                case "Fechar":
                    f.setVisible(false);
                    break;

                case "Abrir": {
                    // Crie um objeto da classe JFileChooser, que permite sel
ecionar local e nome de arquivos
                    JFileChooser j = new JFileChooser("C:");

                    // chamando a função showsOpenDialog para mostrar a caixa
de diálogo salvar
                    int r = j.showOpenDialog(null);

                    // Se o usuário selecionar um arquivo
                    if (r == JFileChooser.APPROVE_OPTION) {

                        // Defina o rótulo para o caminho do diretório seleci
```

```
onado
                        File fi = new File(j.getSelectedFile().getAbsolutePat
h());

                        try {
                            // String
                            String s1 = "", sl = "";

                            // File reader
                            FileReader fr = new FileReader(fi);

                            // Buffered reader
                            BufferedReader br = new BufferedReader(fr);

                            // inicializando sl
                            sl = br.readLine();

                            // selecionando a entrada do arquivo
                            while ((s1 = br.readLine()) != null) {
                                sl = sl + "\n" + s1;
                            }

                            ///Defina o texto
                            t.setText(sl);

                        } catch (Exception evt) {
                            JOptionPane.showMessageDialog(f, evt.getMessage
());
                        }
                    }
                    else {
                        // Se o usuário cancelou a operação
                        JOptionPane.showMessageDialog(f, "Operação cancelad
a");
                    }
                    break;
                }
                case "Salvar": {

                    // Crie um objeto da classe JFileChooser
                    JFileChooser j = new JFileChooser("f:");

                    // Invoque a função showsSaveDialog para mostrar a caixa
de diálogo de salvamento
                    int r = j.showSaveDialog(null);

                    if (r == JFileChooser.APPROVE_OPTION) {

                        // Defina o rótulo para o caminho do diretório seleci
onado
```

```
                        File fi = new File(j.getSelectedFile().getAbsolutePat
h());

                        try {
                            // Criando o file writer
                            FileWriter wr = new FileWriter(fi, false);

                            // criando buffered writer para gravar
                            BufferedWriter w = new BufferedWriter(wr);

                            // escrevendo
                            w.write(t.getText());

                            //forçando gravar todos os dados presentes no flu
xo de saída do arquivo
                            w.flush();

                            //fecha o arquivo
                            w.close();
                        } catch (Exception evt) {
                            JOptionPane.showMessageDialog(f, evt.getMessage
());
                        }
                    }
                    else {
                        // Se o usuário cancelou a operação
                        JOptionPane.showMessageDialog(f, "Operação cancelad
a");
                    }
                    break;
                }
                default:
                    break;
            }
        }

      // Método main da classe
        public static void main(String args[]) {
            MiniEditor e = new MiniEditor();
        }
      }
```

Resultado em operação:

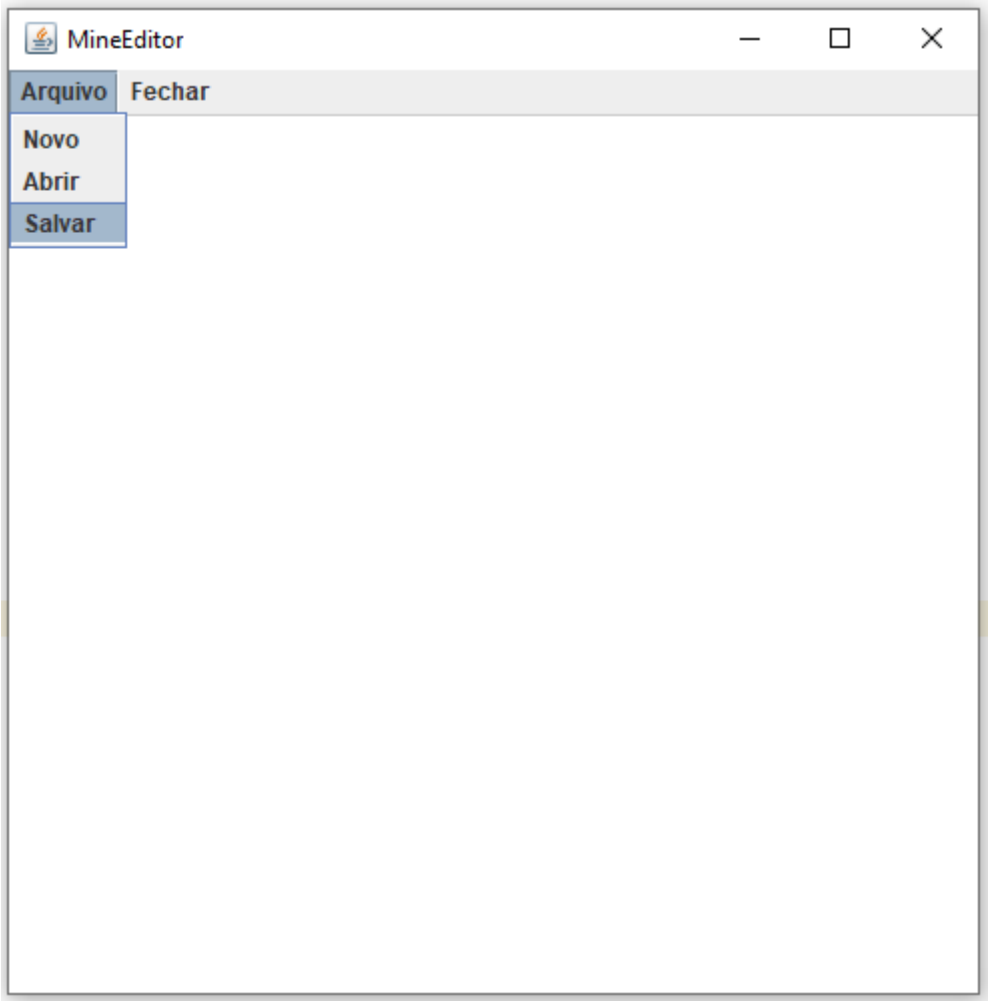

Figura 3 – **MiniEditor** Fonte: Senac EAD (2022)

Lendo arquivos CSV usando a classe BufferedReader

Nesta seção, será realizada a leitura de um arquivo CSV, cuja sigla significa *comma separated values*, ou seja, “valores separados por vírgulas”. Um arquivo CSV é um arquivo simples, que armazena informações como as de planilhas e tabelas, por exemplo.

Para realizar uma leitura dos dados neste tipo de arquivo, ao invés de ler o arquivo de texto por caracteres, pode-se ler uma linha do arquivo de cada vez. No Java, a classe **BufferedReader** do pacote **java.io** pode ser usada para ler

um arquivo CSV. Quando se utiliza essa classe, simplesmente lê-se cada linha do arquivo usando o método **readLine()**. Então, pode-se dividir a linha usando o método **split()** e passando o ponto e vírgula (**;**) como delimitador.

O exemplo a seguir serve de base para mostrar como proceder com a leitura dos arquivos em CSV. Para realizar a leitura dos dados, será simulada a manipulação de um arquivo com as seguintes colunas: disciplina, ano e carga horária.

Arquivo **exemplo.csv**:

```
disciplina;ano;horas
Linguagem Java; 2022; 108
Banco de Dados; 2021; 100
Redes de computadores; 2023; 96
```

Observe a leitura do arquivo, com comentários detalhando os métodos utilizados:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;
import java.util.Arrays;

public class ArquivosCSV {

    public static void main(String[] args) {
        try {
            //criando uma lista para armazenar os dados
            List<List<String>> data = new ArrayList<>();

            //caminho de arquivo
            String file = "C:\\exemplo.csv";
            FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr);

            //lendo linha a linha
            String line = br.readLine();
            while (line != null) {
                //separando as informações através do metodo split dos
dados
                //delimitados em ponto e virgula
                List<String> lineData = Arrays.asList(line.split(";"));
                data.add(lineData);
                line = br.readLine();
            }

            //imprimindo os dados buscados
            for (List<String> list : data) {
                for (String str : list) {
                    System.out.print(str + " ");
                }
                System.out.println();
            }
            //fechando o arquivo
            br.close();
        } catch (Exception e) {
            System.out.print(e);
        }
    }
}
```

O resultado será o seguinte:

```
disciplina ano horas
Linguagem Java  2022  108
Banco de Dados  2021  100
Redes de computadores  2023  96
BUILD SUCCESSFUL (total time: 0 seconds)
```

Escrevendo em arquivos CSV usando a classe FileWriter

Existem muitas bibliotecas que oferecem suporte à gravação em arquivos CSV. No entanto, existe uma maneira simples de gravar em arquivos CSV, assim como em qualquer outro tipo de arquivo, sem usar uma biblioteca.

A maneira mais simples é usar um objeto da classe **FileWriter** e tratar o arquivo CSV como qualquer outro arquivo de texto.

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.List;

public class Gravar {

    public static void main(String[] args) throws IOException {

        FileWriter csvWriter = new FileWriter("C:\\exemplo.csv");
        //criando as informações do cabeçalho
        csvWriter.append("disciplina");
        csvWriter.append(";");
        csvWriter.append("ano");
        csvWriter.append(";");
        csvWriter.append("horas");
        csvWriter.append("\n");

        // Armazenando as informações em um array de string
        List<List<String>> rows = Arrays.asList(
            Arrays.asList("Linguagem Java", "2022", "108"),
            Arrays.asList("Banco de Dados", "2021", "100"),
            Arrays.asList("Redes de computadores", "2023", "96")
        );

        //percorrendo a lista de dados
        for (List<String> rowData : rows) {
            csvWriter.append(String.join(";", rowData));
            csvWriter.append("\n");
        }

        //posicionando no final do arquivo
        csvWriter.flush();
        //fechando o arquivo
        csvWriter.close();

    }

}
```

Ao usar um **FileWriter**, certifique-se sempre de liberar e fechar o fluxo. Isso melhora o desempenho da operação de E/S e indica que não há mais dados a serem gravados no fluxo de saída.

Construa uma aplicação **Swing** (com interface visual) com campos para o usuário informar um nome e uma idade e um botão **Registrar**. Ao clicar nesse botão, grave em um arquivo CSV essas informações, criando esse arquivo, caso ele não exista, ou acrescentando essa informação a ele, caso contrário.

Como foi possível ver ao longo do conteúdo, a leitura e a gravação de arquivos, quando se está programando em Java, são muito simples. O uso do **Stream** permite controlar os fluxos de entrada e saída com facilidade, permitindo ao desenvolvedor várias possibilidades de implementação.