第３章．ファイルを用いたデータ入出力

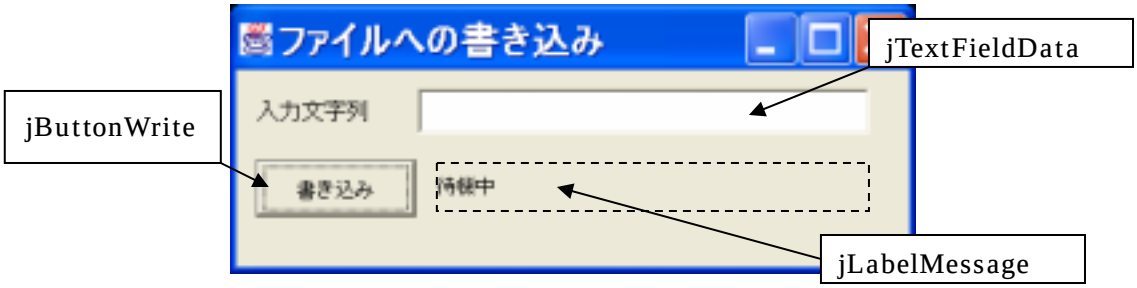
【学習のねらい】

① データをファイルへ出力する（書き込む）方法を学習する。
② データをファイルから入力する（読み込む）方法を学習する。

大量のデータを処理する場合は、ファイルからデータを読み込んでから処理を行います。また、プログラムが終了すると変数に記憶されたデータは消えてしまうので、処理結果を残すためにはファイルに保管しておくことが必要になります。このようにファイルを用いたデータ入出力はコンピュータを使った処理の場合必須の技術です。そこで本章では、Java 言語を用いたファイルによるデータ入出力の仕方を学習しましょう。

３－１　ファイルへの出力１－ファイルへの書き込み方

まず、簡単な例でデータをファイルに書き込む方法を学びましょう。次のようなフレームを作成してください。



そしてボタンクリック時のプログラムを次のように記述してください。

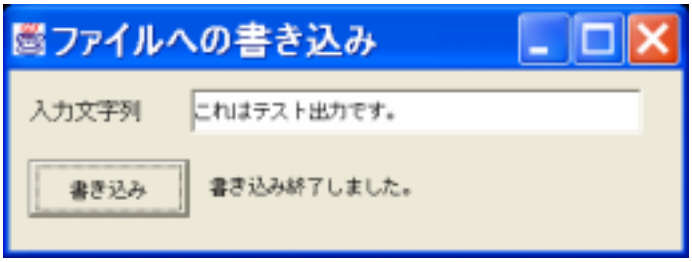
＜プログラム＞

```
  void jButtonWrite_actionPerformed(ActionEvent e) {
    String Data=jTextFieldData.getText();
    try {                                                    ⑥
//    ファイル Test1.txt を出力ストリーム fw として定義する
      FileWriter fw= new FileWriter("Test1.txt");           ①
//    fw への出力時にバッファリング機能をつける。
      BufferedWriter bw= new BufferedWriter(fw);            ②
//    さらに出力時に便利なメソッドを使用可能にする。
      PrintWriter fout = new PrintWriter(bw);               ③
      fout.print("データ入力："+Data);  //ファイルへの書き込み ④
      jLabelMessage.setText("書き込み終了しました。");
      fout.close(); //ファイルを閉じる                       ⑤
    }
    catch(Exception em) {                                    ⑥
      jLabelMessage.setText("エラー発生："+em);
    }
  }
```

加えてプログラムの先頭行に以下の波線部を加えて下さい。以下、ファイルへの入出力を行う場合、これは必ず必要になります。

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
```

見慣れない命令が目につきますが、それらの説明の前にまずは実行して動作を確認してみましょう。このプログラムは、入力文字列欄に入力した文字列を「Test1.txt」というファイルに出力する（書き込む）プログラムです。確認のため、プログラムを起動させ入力文字列欄に「これはテスト出力です。」と入力し、［書き込み］ボタンをクリックして下さい。



すると、「待機中」というメッセージが「書き込み終了しました。」というメッセージに変わります。このとき、プログラム（プロジェクト）と同じフォルダ内に「Test1.txt」というファイルができています。そこで、メモ帳や秀丸エディタなどの適当なテキストエディタで開き、次のように出力されていることを確認してください。

データ入力：これはテスト出力です。

＜Test1.txt＞

＜プログラムの解説＞

０．ストリームについて

Java 言語では入出力データを**ストリーム**という概念で表します。ストリームとは流れという意味です。”流れ”と言われても最初はピンと来ないかもしれませんが、大量のデータをファイルに書き込むあるいは読み出す様は、まさにデータが流れて行くように見えるためストリームと呼ばれるようです。Java 言語では、ファイルへのデータ入出力はストリームの制御（ストリームの行き着く先はどこか？→出力ファイルの指定、ストリームはどこから来るのか→入力ファイルの指定、ストリームをどのように区切って入出力するか？等々）を行うことによって実現されます。Java 言語では、ストリームを制御するために各種のクラスを用意しています（クラスについては第４章で学習します。ここでは、必要な機能をプロパティやメソッドの形で用意している**モノ**と捉えておいてください）。以下、ファイルへの出力に用いられる FileWriter、BufferedWriter そして PrintWriter というクラ

スの用い方を上の例を用いて学習しましょう。番号は、p.33 のプログラム中の番号に対応しています。

① 出力ファイルの定義－FileWriter クラス

```
FileWriter fw= new FileWriter("Test1.txt");
```

によって、FileWriter クラスのオブジェクトを fw という名前で生成しています。一般に、あるクラスのオブジェクトを生成する場合は、

クラス名　オブジェクト名 = **new** クラス名()

と記述します。FileWriter クラスの場合、右辺の FileWriter()の（）内に、出力ファイル名を指定します。一般のクラスの定義や生成については、第４章で学習しますので、ここでは、まだ約束事だと捉えておいて下さい。ともかく、この文によって、ここで扱うストリーム fw がファイル「Test1.txt」に流れ着く**出力ストリーム**として定義されました。

② 出力ストリームをバッファリング可能にする－BufferedWriter クラス

ファイルへデータの出力を行う場合、データ（ストリーム）を１単位（通常はバイト単位）毎に転送すると、フロッピィディスクやハードディスクなどファイル媒体側の受け入れ準備ができるまで待ち時間が発生し効率が悪くなります。そこで、一旦、出力データをメモリにため込み（書き込み）、ファイル側の書き込み準備ができた段階で順次データを書き込む処理が行われます。この処理をバッファリングと言います。

```
BufferedWriter bw= new BufferedWriter(fw);
```

によって、bw はバッファリング機能付きのストリームとなります。

③ サービス機能を付加する－PrintWriter クラス

ファイルへのデータ出力を行う場合、PrintWriter クラスに用意されている print()あるいは println()メソッドを用いるのが普通です。*) そのため、次の文によって出力ストリームを PrintWriter クラスの提供するサービス機能つきのストリームにしました。

```
PrintWriter fout = new PrintWriter(bw);
```

なお、上では説明のために３つに分解して記述したのですが、通常は出力ストリームの定義は次のように１行で記述されます。

```
PrintWriter fout = new PrintWriter(new BufferedWriter(new
                   FileWriter("Test1.txt")));
```

本講義でも以下ではこのように１行で記述することにします。

＊）少し補足すると、FileWriter クラスにもデータを書き込む write()というメソッドが用意されているのですが、実数値を出力できない、またデータの改行を指定できないなど、実用上不便な点が多かったので、PrinWriter クラスの print()および

println()クラスが用意されたのです。

④　ファイルのクローズ

ファイル処理を終了したら、close()メソッドにより、必ずファイルを閉じなければなりません。

⑤　エラー処理－try～catch 文

```
FileWriter fw= new FileWriter("Test1.txt");
```

などのように、出力ストリームを定義する際、何らかの理由で指定したファイルを出力ファイルとして確保できなかった場合、(FileWriter クラスは) IOException という例外(情報）を発行します。Exception（例外）とは、想定通りに行かなかった場合、つまりエラー発生の状態を指すプログラミング用語です。Java 言語では、Exception 情報を発行する事を、”Exception 情報を投げる ”と言います。そして投げられた Exception 情報をキャッチ（**catch**）して適切な処理を行うよう、文法的に義務づけています。一般に、ファイルの入出力を行う場合、

```
try {
  ファイル入出力の処理
}
catch (Exception em) {
  エラーに対応した処理
}
```

という形で処理を記述します。今の場合、エラー処理は

```
catch(Exception em) {
  jLabelMessage.setText("エラー発生："+em);
}
```

となっており、これは、エラー情報を（ラベルに）表示させる処理を行うという意味です。catch 文の()内の em には Exception の情報、つまりエラーメッセージが書き込まれているのです。

少し説明が長くなりましたので、実際に色々な動作を確認することで理解を深めましょう。上のプログラムに以下の波線部を追加して実行してみて下さい。

```
void jButtonWrite_actionPerformed(ActionEvent e) {
  String Data=jTextFieldData.getText();
  try {
    ・・・
    fout.print("データ入力："+Data);  //ファイルへの書き込み
    fout.print(”次の入力");
    jLabelMessage.setText("書き込み終了しました。");
    ・・・
    以下同じ
}
```

先程と同様に文字列入力欄に「これはテスト出力です。」と入力してボタンをクリックすると、ファイル「Test1.txt」の内容は次のようになっているはずです。

データ入力：これはテスト出力です。次の入力

つまり、print()メソッドを連ねると、該当文字列が（横に）続けて出力されます。
それでは、

データ入力：これはテスト出力です。 次の入力

＜Test1.txt＞

というように、改行して出力するにはどうしたらよいでしょうか？ そのためには、最初の print()メソッドを println()メソッドに変更すればよいのです。次のように書き換えて実行してみて下さい。

```
fout.println("データ入力："+Data);  //ファイルへの書き込み
fout.print(”次の入力");
```

このように println()文は()内の文字列を出力後改行する、という意味を持ちます。

【基礎課題 3-1】

上の例では、文字列を採り上げましたが、整数でも同様に出力できます。
上のプログラムに波線部の修正および点線枠の追加を行って実行して下さい。

```
・・・
fout.println("データ入力："+Data);  //ファイルへの書き込み
fout.println("次の入力");
```

```
int a=2,b=5;
fout.println("a="+a+"  b="+b);
fout.println("a+b="+(a+b));
```

```
jLabelMessage.setText("書き込み終了しました。");
・・・
```

そして、「Test1.txt」にどのように出力されるかを確認し、枠内を埋めて下さい。

＜Test1.txt の出力結果＞

データ入力：これはテスト出力です。 次の入力 [ ]

＜解説＞

文字列同士を「＋」で結ぶと、それは連結を意味する、ということはこれまで学習した通りです。しかし、上のプログラムでは整数型と文字列型を「＋」記号で結んでいます。「そんなことが許されるのか？」と首をかしげた人もいることと思いますが、Java 言語ではこのような書き方を認めています。そして結果を確かめてみれば分かるとおり、整数と文字列が連結されて表示されます。例えば次のようなプログラムを考えた場合、

```
int a=2;
String Ans="a="+a;
```

文字列型変数 Ans の値は「a=2」となります。このように、文字列と整数を＋記号で結ぶと、整数が自動的に文字列化され、文字列の連結として処理されます。これは非常に便利なので、Java 言語では頻繁に利用されます。

もちろん、実数についても同様です。例えば

```
int a=2;
String Ans="a/4="+(a/4.0);
```

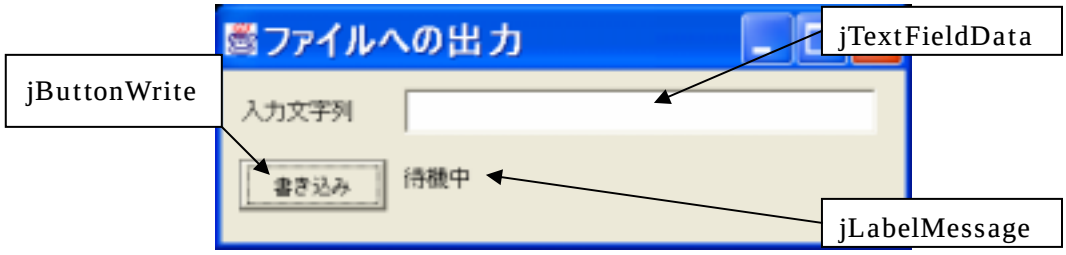
というプログラムを実行すると、文字列型変数 Ans の値は「a/4=0.5」となります。

３－２　ファイルへの出力２－ダイアログによるファイル名の指定

３－１節では、出力ファイル名が固定されているものとして、プログラム中で直接ファイル名を指定していました。しかし、実際には、プログラム実行時にファイル名を指定できた方が便利です。次の課題でその方法を学習しましょう。

【基礎課題 3-2】

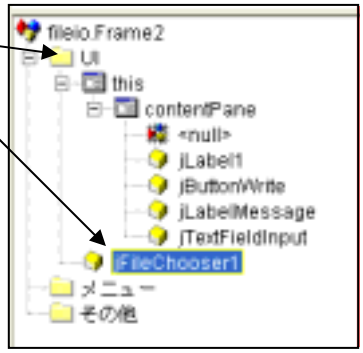
前節と同様、次のようなフレームを作成してください。



次に、コンポーネントパレットから「Swing Containers」タブにある、「JFileChooser」コンポーネントを選択（クリック）して下さい。

選択後、構造ペインの UI フォルダをクリックします（フレーム上に貼り付けるのではないことに注意して下さい）。

すると、JFileChooser コンポーネントが追加されます。



以下の課題では、幾つかのファイルをデータ入出力に用います。そこで、それらファイルを一括して管理できるよう、適当な場所にファイル保管用のフォルダ「IOFile」を作成しておいて下さい。フォルダ名は何でも良いのですが、以下では、ファイル保管用フォルダ名が「IOFile」という名前であるとして説明しています。

そしてボタンクリック時のプログラムを次ページのように作成します。

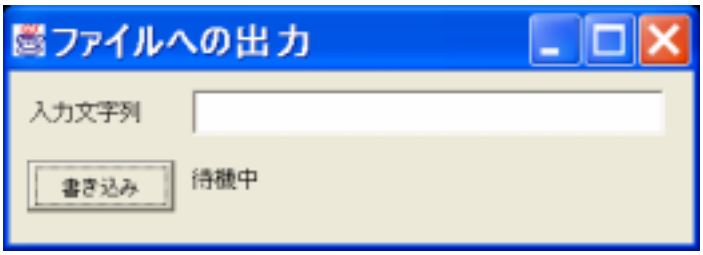
```
  void jButtonWrite_actionPerformed(ActionEvent e) {
    String Data=jTextFieldData.getText();
    try {
//    ダイアログボックスを開く
      jFileChooser1.showOpenDialog(this);                    ①
//    出力ファイル名を指定する
      File FName=jFileChooser1.getSelectedFile();            ②
//    出力ストリームを定義する
      PrintWriter fout=new PrintWriter(new BufferedWriter
                                  (new FileWriter(FName)));  ③
      fout.println("データ入力："+Data);  //ファイルへの書き込み
      jLabelMessage.setText("書き込み終了しました。");
      fout.close(); //ファイルを閉じる
    }
    catch(Exception em) {
      jLabelMessage.setText("エラー発生："+em);
    }
  }
```

なお、プログラムの冒頭に、

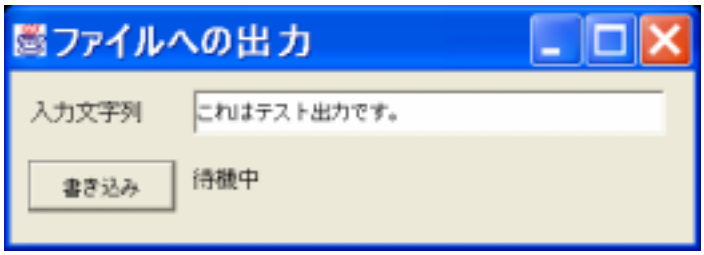
```
      import java.io.*;
```

を追加することを忘れないようにして下さい。ファイル名を指定している①～③以外は前節のプログラムと同じです。その説明は後回しにして、とりあえずプログラムを実行してみましょう。

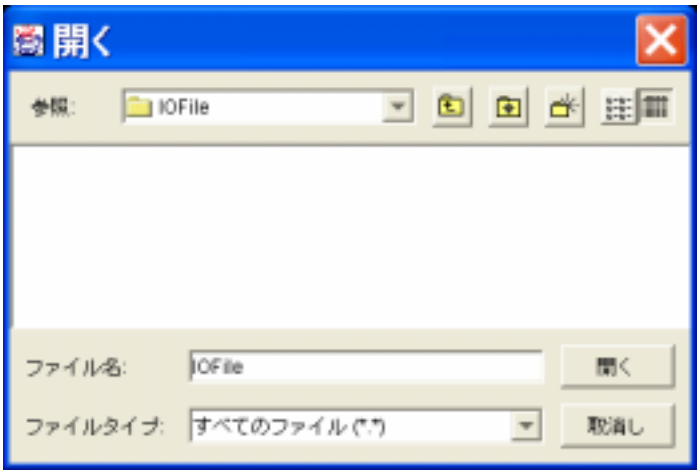
実行後、前節同様次のような起動画面が現れます。



ここで、入力文字列欄に「これはテスト出力です。」と入力し［書き込み］ボタンをクリックすると・・・

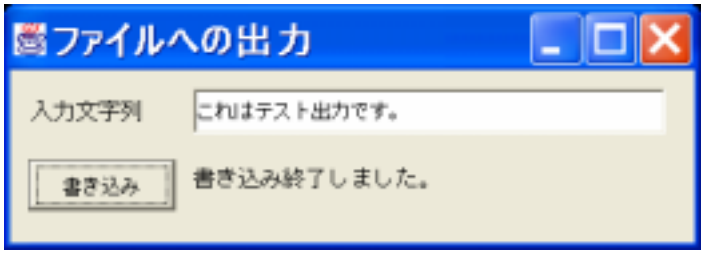


次ページのように出力ファイル指定のためのダイアログボックスが現れます。ここで、次のように、先程作成した「IOFile」フォルダを指定します。



さらに、ファイル名として「output.txt」を入力し、［開く］ボタンをクリックすると・・・

ダイアログボックスは消え、元の画面に戻ります。



この時点で、出力ファイル「output.txt」に入力文字列欄に指定した文字列が書き込まれています。確認のため、［IOFile］フォルダ内の「output.txt」ファイルの中身を適当なエディタでチェックしてみて下さい。次のように書き込まれているはずです。

データ入力：これはテスト出力です。

＜output.txt＞

このように、JFileChooser コンポーネントを用いると、プログラム実行時に任意のファイルを指定することができます。それでは、先程のプログラムの①～③の意味を確認しておきましょう。

＜プログラムの解説＞

① `showOpenDialog()`メソッドにより、ファイルを指定するダイアログボックスが表示されます。`()`内は通常 this を指定するものと理解しておいて下さい。

② `getSelectedFile()`メソッドは、ファイルダイアログボックスで指定したファイル名を返します。ファイル名は「`File`」クラス（型と考えて結構です）の変数（オブジェクト）に代入できます。ここでは、その変数名を「`FName`」としています。

③ 下線部に示したように、今まで`"output.txt"`等と直接ファイル名を指定していた部分をファイル名の変数に置き換えています。

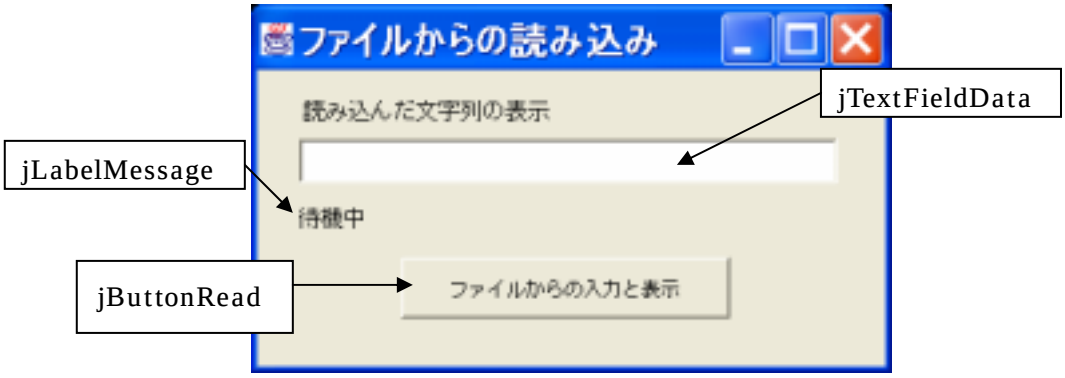
３－３　ファイルからの入力１－ファイルからの読み込み方

さて、本章の後半では、ファイルからのデータ入力の方法を学習することにします。

例として【基礎課題 3-2】で作成した出力ファイル「output.txt」を読み込んでその内容を表示させるプログラムを考えましょう。

【基礎課題 3-3】

次のようなフレームを作成してください。



【基礎課題 3-2】と同様に入力ファイルを実行時にファイルダイアログボックスで指定するので、p.39 と同様に JFileChooser コンポーネントを構造ペインの UI フォルダに追加して下さい。そして、ボタンクリック時のプログラムを次のように記述します。

```
void jButtonRead_actionPerformed(ActionEvent e) {
  String Data;
  try {
    jFileChooser1.showOpenDialog(this);                    ①
    File FName=jFileChooser1.getSelectedFile();
    BufferedReader fin=new BufferedReader                  ②
                          (new FileReader(FName));
    Data=fin.readLine(); //ファイルからの読み込み          ③
    jTextFieldData.setText(Data); //読み込んだデータの表示
    fin.close();
    jLabelMessage.setText("読み込み終了しました。");
  }
  catch (Exception em) {
    jLabelMessage.setText("エラー発生："+em);
  }
}
```

※　プログラムの冒頭に、「**import** java.io.*;」を追加することを忘れないように注意して下さい。

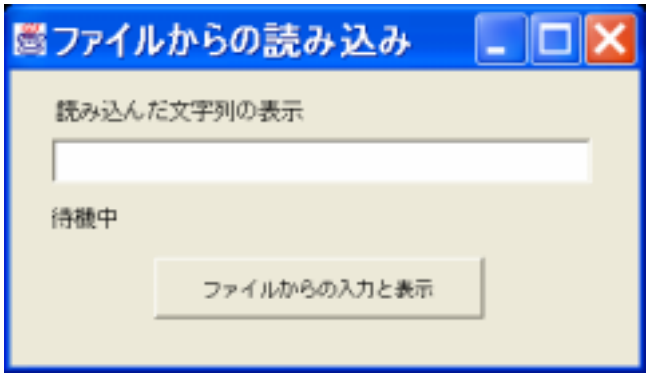
＜プログラムの解説＞

①　ダイアログボックスで指定した（入力）ファイル名を受け取る部分は出力ファイルの場合と同じです。

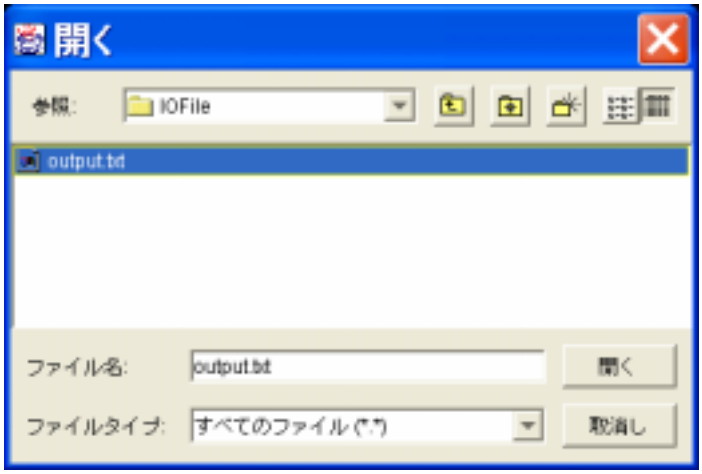
② 入力ストリーム「fin」の指定（定義）です。出力ストリームの場合と比べて PrintWriter クラスに該当する部分がありませんが、それ以外は出力ストリームの場合と同様です。

③ `readLine()`メソッドは、入力ストリーム（今の場合ファイル「FName」）から１行分だけ読み込み、その値を文字列として返します。

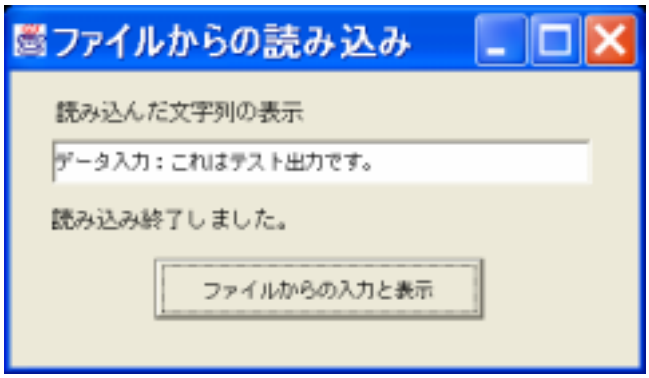
プログラムの内容はほぼ理解できると思いますので早速実行して動作を確認してみましょう。



このプログラムを起動し、[ファイルからの入力と表示] ボタンをクリックします。



すると、ファイル指定のためのダイアログボックスが現れるので、ここで、【基礎課題 3-2】で作成した「output.txt」を指定します。ファイル名の指定後、[開く] ボタンをクリックすると・・



元の画面に戻り、ファイル「output.txt」の内容が表示されます。

このプログラムを用いればどの様なファイルでも、その内容を表示させることができます。ただし、最初の１行分だけですが・・・。

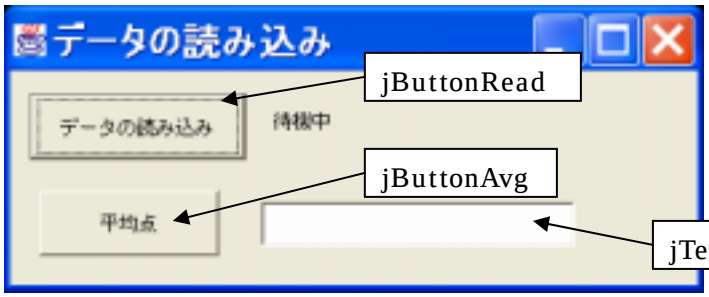
【基礎課題 3-4】 ー　複数行のデータの読み込み

今度は、複数行のデータをファイルから読み込んでみましょう。入力ファイルとして、HP の該当部分に掲載している「input.txt」ファイルをダウンロードし、前節で作成したフォルダ「IOFile」にコピーして下さい。このファイルには、次のように、あるテストの５人分の得点がデータとして入力されています。今、このファイルからデータを読み込み、このテストの平均点を表示するプログラムを考えましょう。

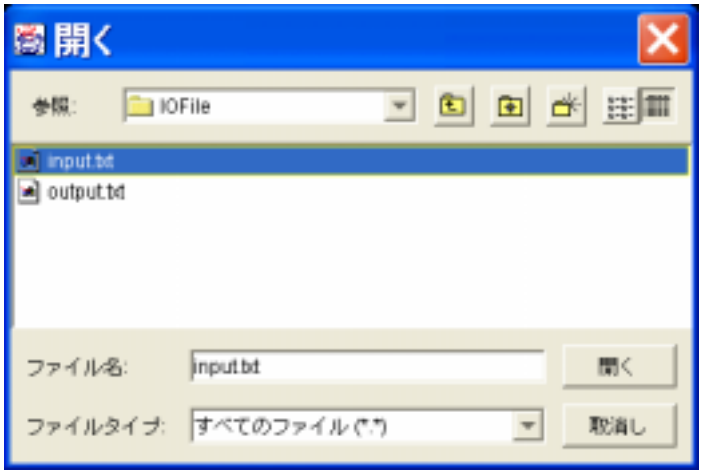
＜input.txt＞

```
55
60
92
38
71
```

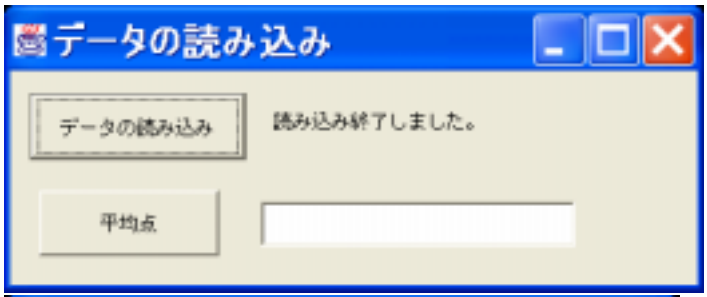
作成するプログラムの動作内容は次の通りです。



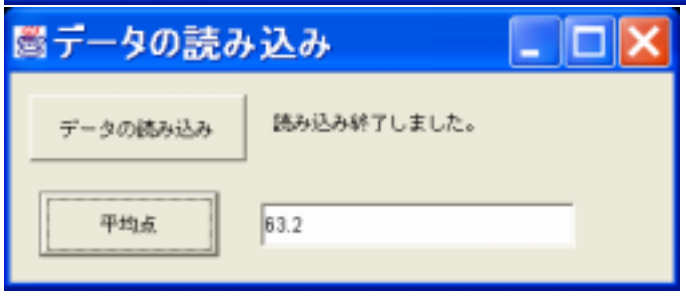
プログラムを起動すると左のような画面が現れます。
ここで、[データの読み込み]ボタンをクリックすると・・・



入力ファイルを指定するダイアログボックスが開きますので、ここで、今ダウンロードしたファイル「input.txt」を指定します。その後［開く］ボタンをクリックすると・・・



元の画面に戻り、表示が「読み込み終了しました。」に変わります。この時点で入力データの読み込みは完了しています。



続いて［平均点］ボタンをクリックすると、テストの平均点が表示されます。

それではプログラムを作成しましょう。今の場合、簡単のためにデータの個数は５つであることが分かっていることにしましょう。作成に当たってポイントとなるのは次の点です。

① ５つのデータ（得点）を保管するために、大きさが５の配列型変数を用意する。

② ファイルから１行ずつデータを読み込む方法は【基礎課題 3-3】で分かっているので、５つのデータを読み込むには、それを５回繰り返せば良い。

③ 今の場合、読み込んだデータを保管する配列型変数は、ボタン［データの読み込み］とボタン［平均点］の２つのイベントハンドラ（メソッド）で共有（参照）するので、配列型変数の宣言は、メソッドの外で行わなければならない。なぜならメソッドの中で宣言すると、そのメソッド内でしか参照できないからである。→前期のテキスト「6-12 グローバル変数とローカル変数」参照。

以上を念頭に置いてプログラムを記述すると次のようになります。

```
  int Tokuten[]=new int[5];  //得点を保管する配列          } ①
  int Num;  //データの個数

  void jButtonRead_actionPerformed(ActionEvent e) {
    String Data;
    try {
      jFileChooser1.showOpenDialog(this);
      File FName=jFileChooser1.getSelectedFile();
      BufferedReader fin=new BufferedReader
                              (new FileReader(FName));
//    ファイルからの読み込み
      Num=5; //データ個数の設定
      for (int i=0;i<Num;i++) {                          } ②
        Data=fin.readLine();
        Tokuten[i]=Integer.parseInt(Data);
      }
      jLabelMessage.setText("読み込み終了しました。");
      fin.close();
    }
    catch (Exception em) {
      jLabelMessage.setText("エラー発生："+em);
    }
  }

  void jButtonAvg_actionPerformed(ActionEvent e) {
//    平均点の計算
    int sum=0;
    for (int i=0;i<Num;i++) {                            } 得点の合計の計算
      sum=sum+Tokuten[i];
    }
    double Avg= sum/(double)(Num);          ③
    jTextFieldAvg.setText(String.valueOf(Avg));
  }
```

＜プログラムの解説＞

① ５つのデータを保管する整数型配列変数 **Tokuten** とデータの個数を保管する整数型変数 **Num** を宣言します。場所は、メソッド（イベントハンドラ）の外側ならどこでも良いのですが、ここでは、分かりやすく［データの読み込み］ボタンクリック時のイベントハンドラの上に置きました。

② データの個数分だけ（１行ずつ）データを読み込み、それを Tokuten[i]に順次代入しています。データを読み込んだ段階では文字列型なので、Tokuten[i]に代入する際には整数型への変換が必要であることに注意して下さい。

③ ここで平均点を計算しています。Java 言語のルールにより、「整数／整数」は小数点以下が切り捨てられ整数になる、という点に注意して下さい。そのため、分母の Num を実数型に型キャスト（変換）しています。型キャストについては前期のテキスト p.72 を参照して下さい。

プログラム作成後は実行して動作を確認して下さい。

３－４　ファイルからの入力２－連続したデータ読み込み

【基礎課題 3-4】では予め入力データの個数がわかっているものとしていましたが、実際にはデータの個数が分からない状態で入力処理を行う場合が少なくありません。そこで次の課題で、ファイル中のデータの個数が予め分かっていない場合に対応できるように（【基礎課題 3-4】のプログラムを）改善しましょう。

【基礎課題 3-5】

【基礎課題 3-4】のプログラムにおいて、配列の大きさを下線部の様に修正し、データ読み込み部分を枠線部の様に修正して下さい。それ以外は以前の通りです。

```
int Tokuten[]=new int[100];  //得点を保管する配列          ①
int Num; //データの個数

void jButtonRead_actionPerformed(ActionEvent e) {
  String Data;
  try {
    jFileChooser1.showOpenDialog(this);
    File FName=jFileChooser1.getSelectedFile();
    BufferedReader fin=new BufferedReader
                              (new FileReader(FName));
//    ファイルからの読み込み
    int i=0;
    while ((Data=fin.readLine())!=null) {
      Tokuten[i]=Integer.parseInt(Data);
      i++;
    }
    Num=i; //データの個数の保管                          ②
    jLabelMessage.setText("読み込み終了しました。");
    fin.close();
  }
  catch (Exception em) {
    jLabelMessage.setText("エラー発生："+em);
  }
}
```

＜プログラムの解説＞

① 予めデータの個数が決まっていないので、予想される最大個数程度を配列の大きさに指定しておきます。ここでは 100 にしています。

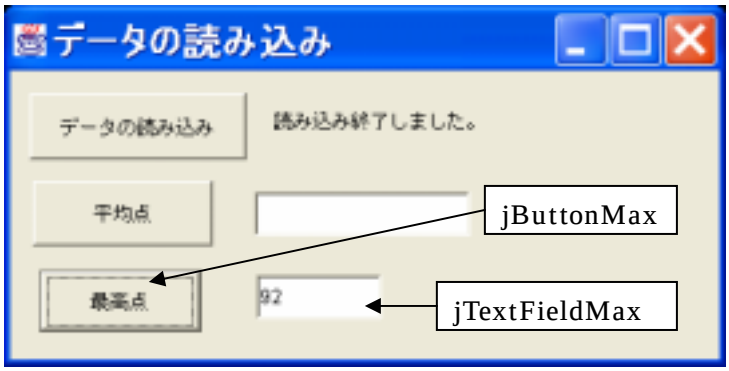
② readLine()メソッドによって、ファイル中の最終データの**次を**読み込んだ場合、そこには何もないので、「null」という特別の値になります。null は「空っぽ」という意味です。「**while ((Data=fin.readLine())!=null)**」では、入力ストリームから

読み込んだ値を Data に代入し、その値が「null」ではない場合、つまりデータが存在する限り読み込み続ける、という条件指定になっています。なお、while ループを回る回数は「データ個数＋１」なのですが、カウンタ i が 0 から始まっているため、最終的に i 値がそのままデータの個数になっています。

上の様に修正後、プログラムを実行し、前節同様きちんと動作することを確認して下さい。

【基礎課題 3-6】

【基礎課題 3-5】のプログラムに、以下のように最高点を求め表示するボタンを加えて下さい。



入力ファイルを指定してデータを読み込んだ後、［最高点］ボタンをクリックすると、最高点が表示される。

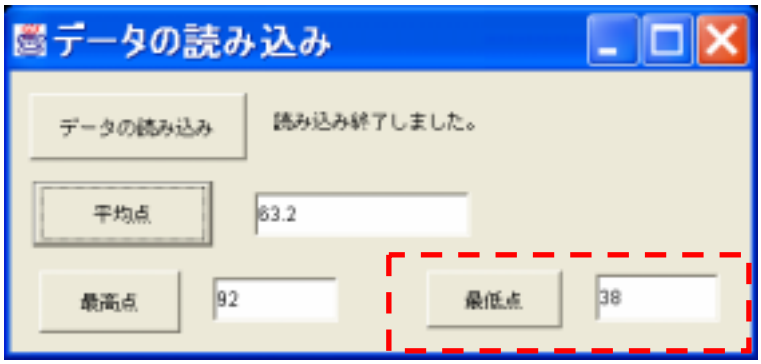
下の枠内を埋めて最高点を求めるプログラムを完成させて下さい。

```
void jButtonMax_actionPerformed(ActionEvent e) {
  int Max=Tokuten[0];
  for(int i=1; i<Num;i++) {
```

```
  }
  jTextFieldMax.setText(String.valueOf(Max));
}
```

【応用課題 3-A】

【基礎課題 3-6】に今度は、最低点を求める処理を加えて下さい。



【応用課題 3-B】

HP の該当部分より、あるテストの得点の入ったファイル「score.txt」をダウンロードして下さい。そして、【応用課題 3-A】のプログラムを用いて、このテスト得点のデータの平均点および、最高点、最低点を求めて下さい。

<score.txt>

55
90
85
38
80
・・・

※　このデータは 100 名以上です。配列の大きさを 150 に変更して下さい。

次に、左のように、［データの読み込み］ボタンをクリックした時に受験者人数を表示するように拡張して下さい。

【応用課題 3-C】

さらに【応用課題 3-B】を拡張しましょう。今、テストの得点が 50 点以上であれば合格であるとしましょう。そこで、「score.txt」から得点を読み込んで、［合格者］ボタンをクリックすると、合格者数を表示する部分を加えて下さい。