第９章．連結リスト

【学習のねらい】

① 連結リストの概念を理解する。
② 連結リストに関する基本操作（挿入、削除）のプログラミングを学習する。

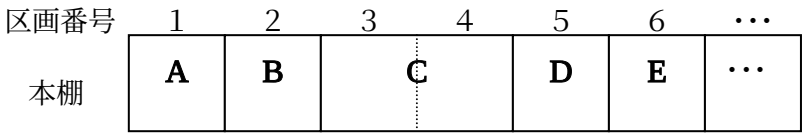
第５章（p.84～85）で、オブジェクト名（第５章で用いた Meibo1 など）は、通常の変数と違って、当該オブジェクトのメモリ上の記憶場所を保管する ” 参照型変数 ” であることを説明しました。本章では、この ” 参照 ” という概念（機能）を用いて、連結リストというデータ構造を学習します。参照を利用すると問題に応じて様々なデータ構造を定義することができます。連結リストとはその中でも最も単純なものですが、その一方で、（十分に）専門的な学習内容と言えます。もっとも、それは理解が困難だと言うことではありません。これまでの知識を基にすればきちんと理解できるはずです。ただ、その際、頭の中で考えるだけではなく、実地のプログラミングでその処理の流れがどのように進んでいるかを確認する（理解する）、という姿勢がこれまでにも増して重要になります。本章でもその点を重視して説明を加えています。どうかそのつもりで、説明を読み飛ばすことなく、じっくりと動作内容あるいは処理の流れを確認しながら学習して行ってください。

それでは、まず、” 参照 ” の復習から始めましょう。

９－１　参照について（アドレス）

これまで、プログラム作成の際には、様々なデータを保管するために（整数型や実数型などデータの種類に応じた）変数を用いて来ました。では、これら変数はコンピュータ上でどのように保管されているのでしょう？

一言で言えば、変数（の値）は、コンピュータのメモリ上に保管されています。それは、本棚に本をしまう場合に似ています。今の場合、本棚がメモリで、その中に入れる本が変数（の値）ということになります。この例えを利用してもう少し説明を続けましょう。下の様に、本を置くスペースがきちんと区画化された本棚を考えます（ブックエンドで区切ればよいでしょう）。そして各区画には番号が振られているものとします。ここに、端から順番に本 A、本 B、･･･を入れて行きます。本によってはその厚さのため本 C の様に 2 区画必要とする場合もあるでしょう。



次に、この本棚から、誰かに頼んで、例えば本 B を取ってきてもらう場合を想定します。その際には、「区画番号 2 にある本を持ってきて！」と言えば間違いありませんね。本 C の

場合なら、「区画番号３～４を占めている本を持ってきて！」となります。

少し回りくどくなってきて退屈してきたと思いますから、ここで本題のコンピュータ・メモリに戻りましょう。メモリの場合、上の区画番号をアドレス（番地）と言います。アドレスとはまさに住所（変数の所在地）のことですが、コンピュータ・メモリ上での住所は、上で説明した様な区画番号に他なりません。そこで、本棚と本ではなく、アドレスと変数、という言葉で前頁の図を説明すると「アドレス２に入っているのは変数 B（の値）である。」、あるいは、「アドレス３からアドレス４までを占めているのは、変数 C（の値）である。」というような言い方になります。これで、コンピュータ用語らしくなりました。

以上で、アドレスというものの意味が分かって来たと思います。プログラムで使用する変数は全てメモリ上に保管され、どの変数がどこにあるかはアドレスで指定されるのです。ここで、ようやくメインテーマの ” 参照 ” に話を進めることができます。実は、Java 言語ではこのアドレスのことを ” 参照 ” と呼びます。そして参照が保管される変数（オブジェクト名や配列名など）を ” 参照型変数 ” と呼びます。以下、説明の都合上、参照の値、つまりアドレスのことを参照値、そして参照型変数を参照と呼ぶことにします。

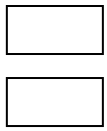
以上を念頭に置いて、もう一度第５章（p.84～85）で説明したオブジェクトの代入、つまり参照の代入（の意味）について振り返ってみましょう。今、Meibo というクラスのオブジェクト A,B を宣言・生成し、オブジェクト B をオブジェクト A に代入するという処理を行った場合、どのような動作が起こるのかを（改めて）確認してみます。

＜メモリ上の動作＞

① オブジェクト A,B の宣言

```
Meibo A,B;
```

→

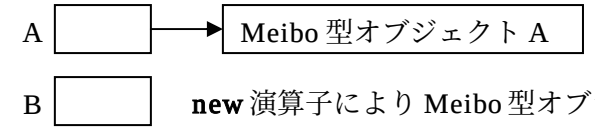


２つの保管領域を確保し、その保管領域を A および B と命名

② オブジェクト A の生成

```
A=new Meibo();
```

→



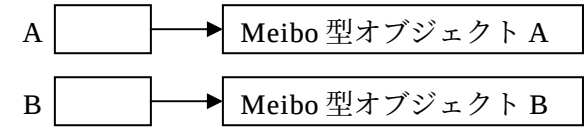
new 演算子により Meibo 型オブジェクトをメモリ上に生成し、その **” 参照値 ”**（アドレス）を A に代入する。オブジェクトの中身が保管されるのではない事に注意。

※　オブジェクトの参照値が代入されると、当該オブジェクトを参照する事が可能になります。この状態を、上の様に参照先を→で指し示す、という表記で表現します。

③ オブジェクト B の生成

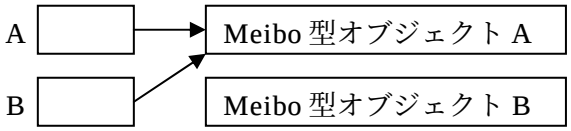
```
B=new Meibo();
```

→



④　B=A の実行

B=A;　→



参照（型変数）B にも、A が参照するオブジェクトの ”**参照値**”が代入されます。したがって、「Meibo 型オブジェクト B」はどこからも参照されなくなりました。

上の④の時点で、参照（型変数）A、B は、共通のオブジェクトを参照しています。

それでは、連結リストの学習に進みましょう。以下の学習に際しては次の点を頭に良く入れておいて下さい。

①　オブジェクト名は参照型変数である。

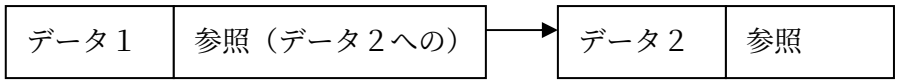
②　参照（値）とは、（当該オブジェクトが保管されている）アドレスのことである。

③　オブジェクトの参照（値）が代入されると、当該オブジェクトを参照する事が可能になる。その際、（以下では）上のような→記号で参照先を指す記法を用いる。

９－２　連結リストとは？

それでは、回りくどいたとえ話はやめてストレートに連結リストの定義を述べることから始めます。（その意味はともかくとして）適当なデータ部と、あるオブジェクトへの参照からなるオブジェクトを考えます。そのオブジェクトを模式的に次のように表しましょう。

そして、この参照が次に続くオブジェクトのアドレスを指すように設定します。すると、下の様に、２つのオブジェクトを参照でつなげることができます。



さらに、データ２の（後ろの）参照が次のデータ３（を有するオブジェクト）のアドレスを指せば、・・・という操作を繰り返して行けば、一連のデータの連なり、つまりリストを表現できます。このように（参照によって）つながれたリストのことを**連結リスト**と呼びます。ヒモによってつながれたカードを連想すれば良いでしょう。

この定義に特に問題はないと思います。でも、「なぜそのようなデータ構造を定義するの？」と疑問に思う人は多いかもしれません。そしてさらに、「リストを表現するなら、配列で十分では？」と思うかもしれません。確かに配列を用いればリストを表現する事は可能です。しかし、配列中のある要素にデータを挿入しようとする場合、第２章で学習した通り、少し手間がかかります。この点を振り返っておきましょう。

今、次のように要素数が５つの配列 A を考えます。

	［1］	［2］	［3］	［4］	［5］
A	データ１	データ２	データ３	データ４	データ５

ここに、A[4]にデータ X を挿入するものとします。すると、2-4 節で学習した通り、以下のような手順を踏まねばなりません。

１　新たに A[6]を用意する。

	［1］	［2］	［3］	［4］	［5］	［6］
A	データ１	データ２	データ３	データ４	データ５	

２　「データ５」を A[6]へ移動する。

	［1］	［2］	［3］	［4］	［5］	［6］
A	データ１	データ２	データ３	データ４		データ５

３　「データ４」を A[5]へ移動する。

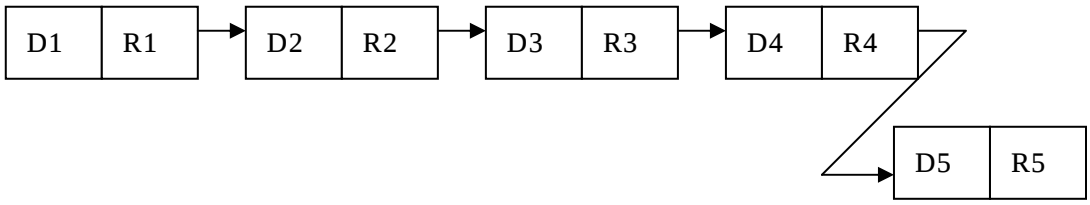
	［1］	［2］	［3］	［4］	［5］	［6］
A	データ１	データ２	データ３		データ４	データ５

４　「データ X」を A[4]に代入する。

	［1］	［2］	［3］	［4］	［5］	［6］
A	データ１	データ２	データ３	データ X	データ４	データ５

削除の場合も同様の手続きを踏まねばなりません。つまり、配列の場合、挿入あるいは削除する要素以降の要素をずらさなければならない、という点に注意してください。

今度は、同様の操作を下のような５つの要素を持った連結リストで考えてみましょう。

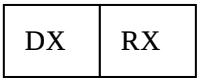


ここに、データ１は D1、参照１は R1 などと略記しています。

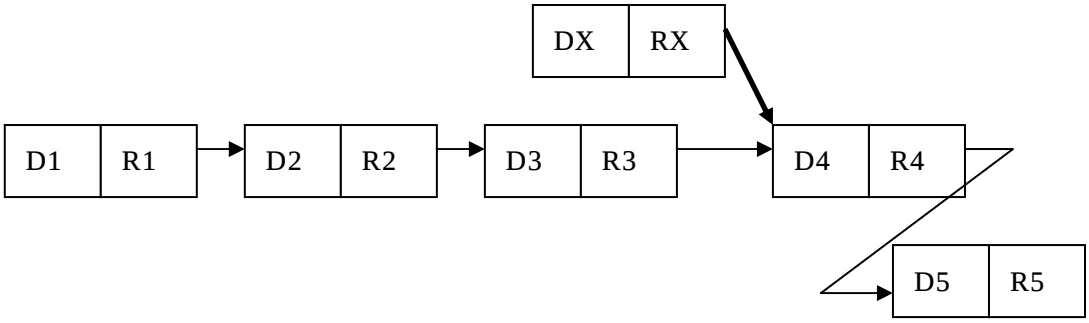
補足　連結リストでは、一つ一つの要素を**セル**と呼びます。セルはそれぞれメモリ上に配置されているのですが、メモリ上のどのアドレスに配置されていても構いません。参照によってつなぐことが出来るからです。一方、配列の場合、要素番号（添え字）順に連続してアドレスに配置されています。

さて、ここで、3 番目のセルの次にデータ X（DX）を持ったセルを挿入するには、次のような手順を踏みます。

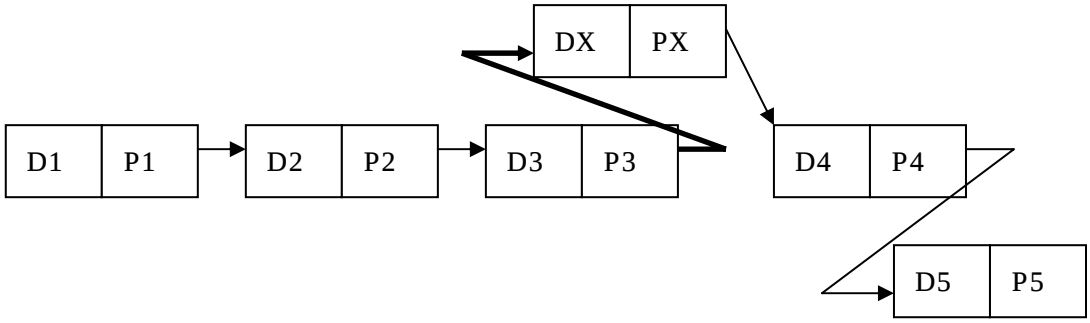
１　データ X のセルを生成する。



２　RX が D4 のセルを指すように変更する。



３　R3 が DX のセルを指すように変更する。



これで、挿入操作は完了しました。挿入する手前の D3 セルの参照を変更した以外は何も変更していない事に注目してください。このように、連結リストは新たなデータの挿入が容易に行えます。削除も同様です。

ここで、配列との比較をもう少しだけ続けましょう。配列の場合、少なくともプログラム実行時には、その大きさ（要素数）を決めておかねばなりません。ところが、連結リストでは、予めデータ数を決めておく必要はなく、幾つでも加えることが可能です。これらの点に注目すると、たとえて言うなら、配列は通常の（綴じた）ノートのようなものです。予め枚数（配列で言えば要素数）が決まっているので、それを越えて記入することができません。そして、あるページに挿入しようとしても、それがすでに書き込まれている場合、そこに書いているものを消してから、書き込むことになります。すでに書いてある内容を保持したい場合は、さらに次のページを消してそこに複写する、といった操作をページ終了まで繰り返すことになります（もちろん実際には、そうまでする人はいませんが・・・）。

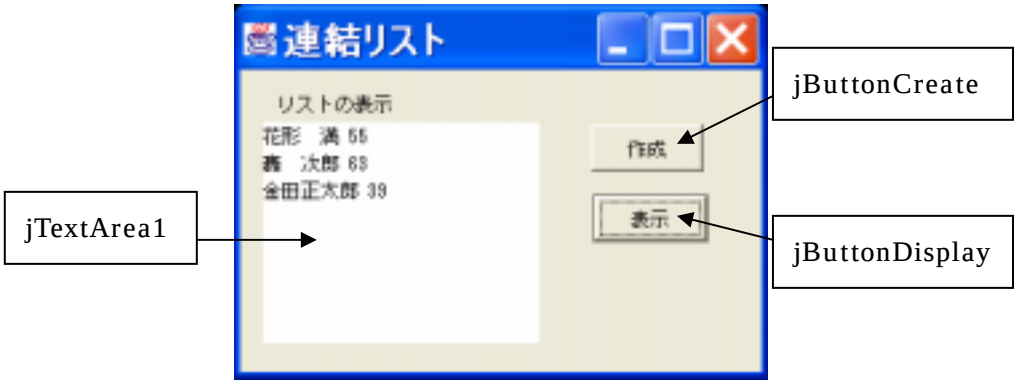
これに対して（みなも使っている）ルーズリーフを思い浮かべてください。この場合は、あるページに 1 枚挿入したければ、直接 1 ページだけ差し込む（挿入する）ことができます。他のファイル（ページ）には何も影響しません。こうして何枚でも増やすことができます。そして削除したい場合は該当ページを抜きとるだけで済みます。このルーズリーフのような、データの挿入や削除などがしやすい ” 融通の利く ” データ構造が連結リストだと捉えておいてください。そして、（正確な対応ではないのですが）「連結リストとはルーズリーフのようなものだ。」と頭に描きながら以下の節を学習すると理解しやすいと思います。

さて、また前置きが長くなってしまいました。以下で、連結リストを作成する方法を学んで行きます。理解すべきポイントは、” 参照によるセルのつなぎ方 ” です。それを念頭に置いて次節に進んで下さい。

９－３　連結リストの作成

連結リスト作成の練習として、次のようなプログラムを作成しましょう。動作内容は次の通りです。

プログラムを起動し［作成］ボタンをクリックすると連結リストを作成し、次に［表示］ボタンをクリックすると、その全要素（氏名とテストの得点）が表示されます。



それでは、プログラムの作成にとりかかりましょう。

＜Cell（セル）クラスの定義＞

このデータは、氏名とテストの得点をフィールドとして持つクラスで表現できます。さらに、そのフィールドに次のデータ（オブジェクト）を指す参照を項目として加えると･･･、それは前節で説明した（連結リストの）セルになりますね。そこで、次のように（セルを表す）クラス **Cell** を定義しましょう。（新規アプリケーションを作成後、次のクラスを新規に作成（定義）して下さい）

```
class Cell {
  private String Name;
  private int Tokuten;
  public Cell Next; //次のセルを指す参照の定義

  public Cell(String Shimei,int Ten) {
    Name=Shimei;
    Tokuten=Ten;
  }
  public String getName() {
    return Name;
  }
  public int getTokuten() {
    return Tokuten;
  }
}
```

※　クラス Cell のフィールド Next は Cell 型クラスです。すると、Cell の定義中に Cell 自身を参照しているので、第８章で学習した再帰的定義になっています。

＜ボタン［作成］＞

ボタン［作成］クリック時のプログラムは次の通りです。大まかな意味は分かると思いますが、処理の流れについては p.148 を参照してください。なお、リストの先頭セルを指す参照「Header」と今注目しているセルの一つ前のセルを指す参照「Previous」をグローバル変数として定義しておきます。なぜ、一つ前のセルを指す参照が必要になるかは、p.148 の＜処理の流れ＞をみれば分かります。

＜［作成］ボタン＞

```
Cell Header,Previous;

void jButtonCreate_actionPerformed(ActionEvent e) {
  Cell Temp;
  //セル１を作る
  Temp = new Cell("花形　満",55); //セル１を生成
  Temp.Next=null; //ひとまず終端にしておく
  Header=Temp; //ヘッダがセル１を指すようにする
  Previous=Temp; //（次の為に）一つ前のセルを指す参照を用意
  //セル２を作る
  Temp = new Cell("轟　次郎",63); //セル２を生成
  Temp.Next=null; //ひとまず終端にしておく
  Previous.Next=Temp; //セル１がセル２を指すようにする
  Previous=Temp; //（次の為に）一つ前のセルを指す参照を用意
  //セル３を作る
  Temp = new Cell("金田正太郎",39); //セル３を生成
  Temp.Next=null; //今の場合、ここがセルの終端
  Previous.Next=Temp; //セル２がセル３を指すようにする
  Previous=Temp; //（次の為に）一つ前のセルを指す参照を用意
}
```

※ **null** はどこも指していない事を意味する参照の値です。リストの終端セルの参照は（次がないので）どこも指しません。そこで、ここでは、リストの終端を表すために用いています。

＜ボタン［表示］＞

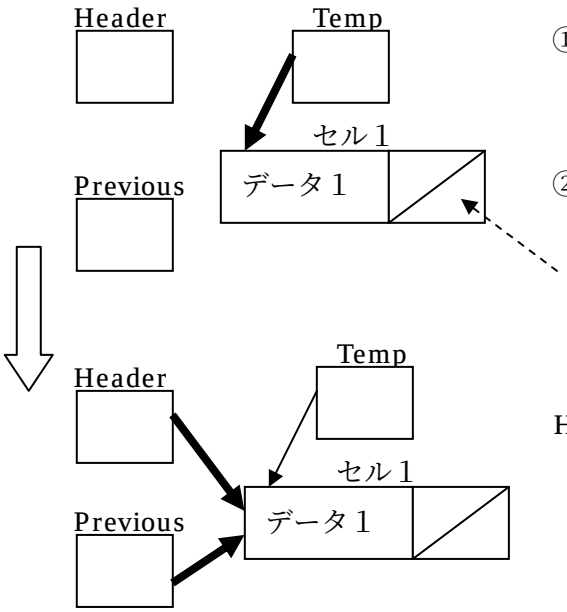
プログラムは以下の通りです。セルの終端を見つけるまで表示を繰り返すという処理内容が理解できるでしょうか？やはり処理の流れを p.150 にまとめておきましたので、下のプログラムと見比べながら、特に Pos の参照先の変化を確認してください。

```
void jButtonDisplay_actionPerformed(ActionEvent e) {
  String Data=""; //空っぽの文字列を用意
  Cell Pos; //操作対象となるセルを指す参照の用意
  Pos=Header; //セルを指す参照をリストの先頭に向ける
  while (Pos!=null) {  //セルの終端でない限り繰り返す
    Data=Data+Pos.getName()+" "+Pos.getTokuten()+"¥n";
    Pos=Pos.Next;  //参照を次のセルに向ける
  }
  jTextArea1.setText(Data);
}
```

＜ボタン［作成］の処理の流れ＞

※　プログラムと見比べながら処理の流れを確認してください。

１．セル１の作成



＜作成処理＞

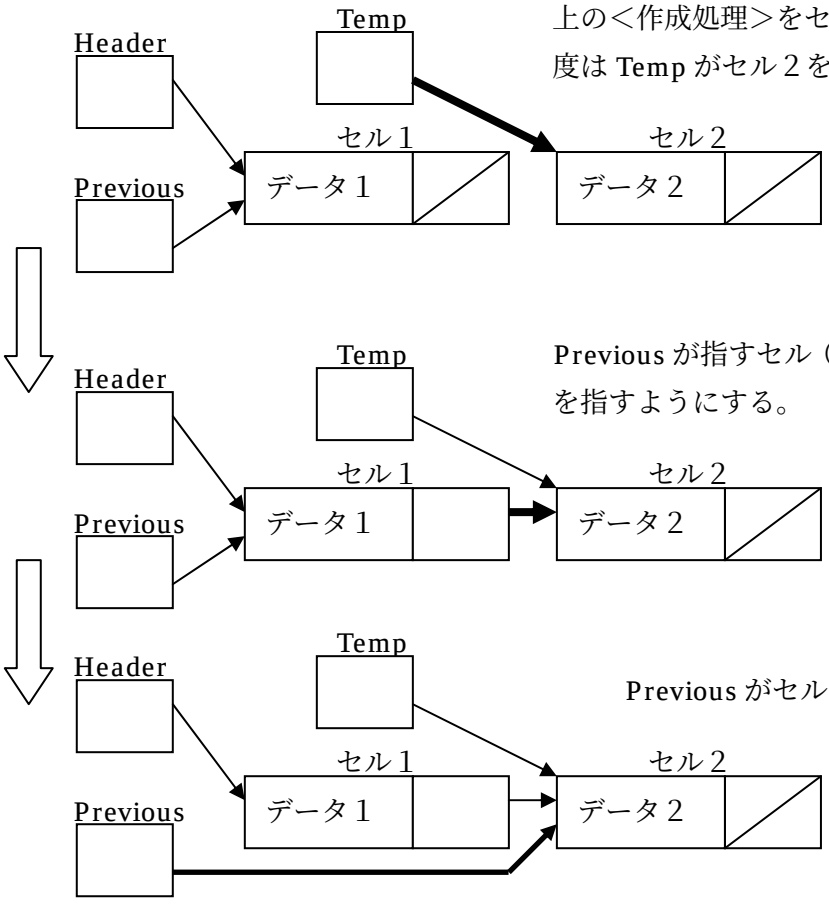
① Cell クラスのオブジェクト用の記憶領域をメモリ上に確保する。さらに参照 Temp がそのアドレスを指すようにする。

② セル 1 の参照 Next の値を **null** にする。

※　参照がどこも指していない状態をこのような斜線で表します。

Header と Previous が、セル１を指すようにする。

２．セル２の作成



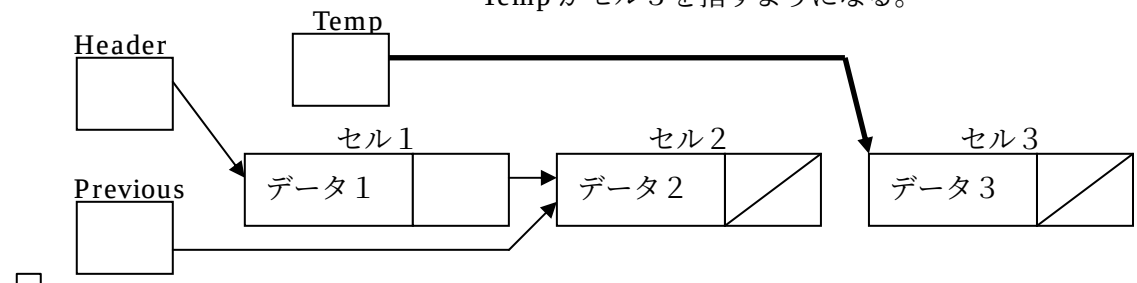
上の＜作成処理＞をセル２について行う。すると今度は Temp がセル２を指すようになる。

Previous が指すセル（セル 1）の参照 Next がセル２を指すようにする。

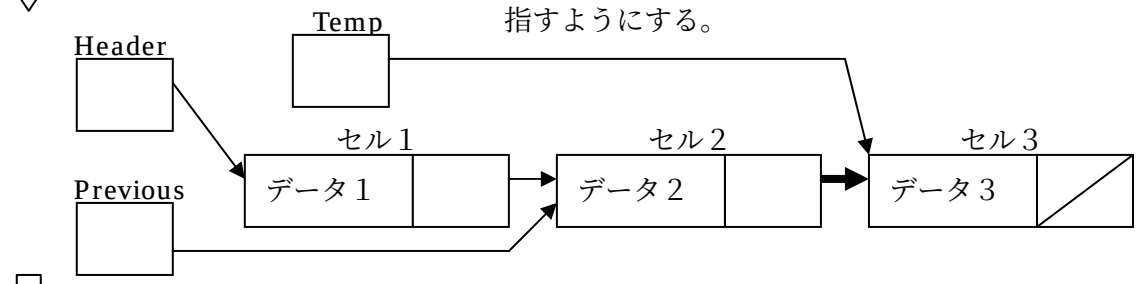
Previous がセル２を指すようにする。

3．セル 3 の作成

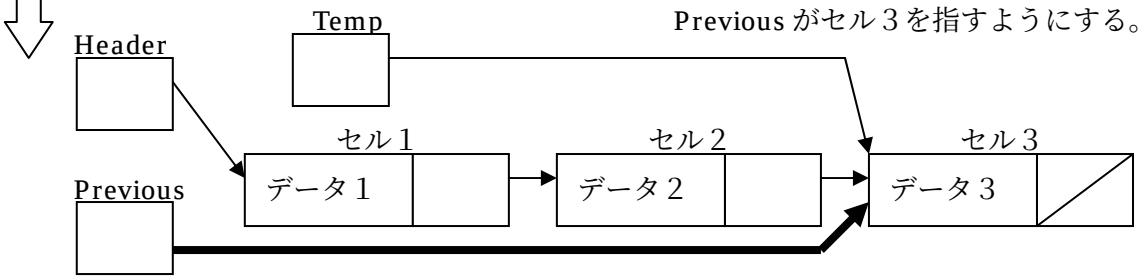
＜作成処理＞をセル 3 について行う。すると今度は Temp がセル 3 を指すようになる。



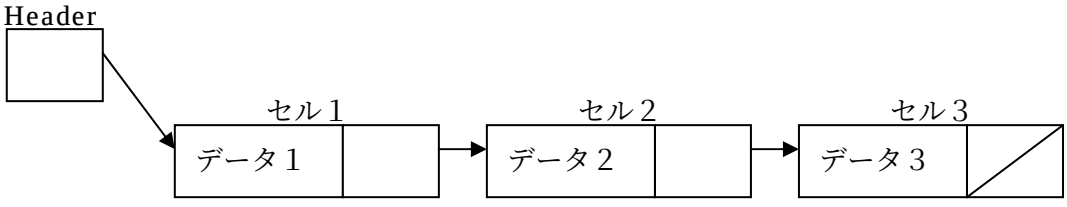
Previous が指すセル（セル 2）の参照 Next がセル 3 を指すようにする。



Previous がセル 3 を指すようにする。



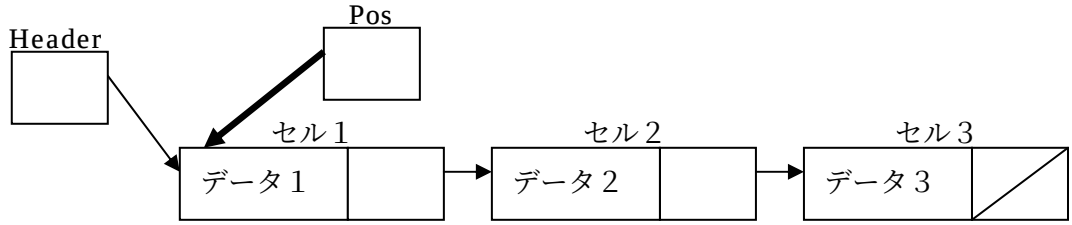
こうして、最終的に次のように、Header からセル 3 まで、参照を通じてたどれる連結リストが出来上がりました。



連結リストでは、セルの先頭を指定するために、Header（先頭という意味）という参照を用意しておきます。そして Header さえ与えられれば、あとはセルの終端まで（参照を通じて）辿れます。

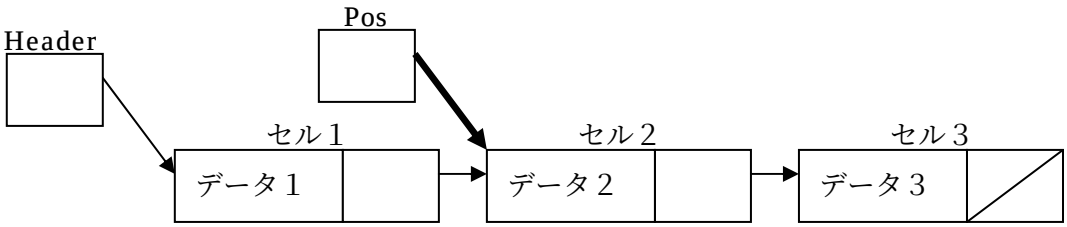
＜ボタン［表示］の処理の流れ＞

1．参照 Pos を Header が指しているセル、つまりセル１に向ける。



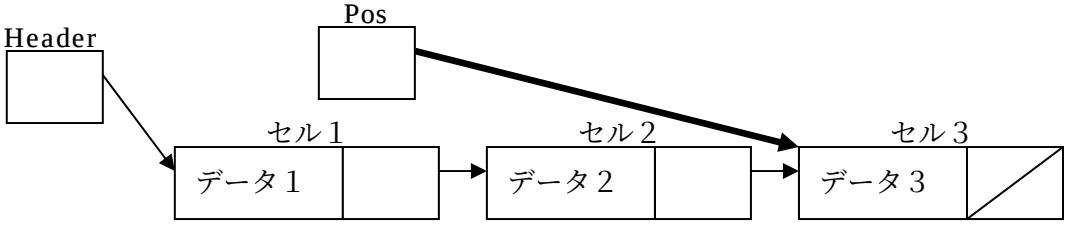
2．Pos は **null** ではないので、セル 1 のデータを文字列変数 Data に連結する。

3．Pos ポインタを、セル１の参照 Next が指しているセル、つまり、セル２に向ける。



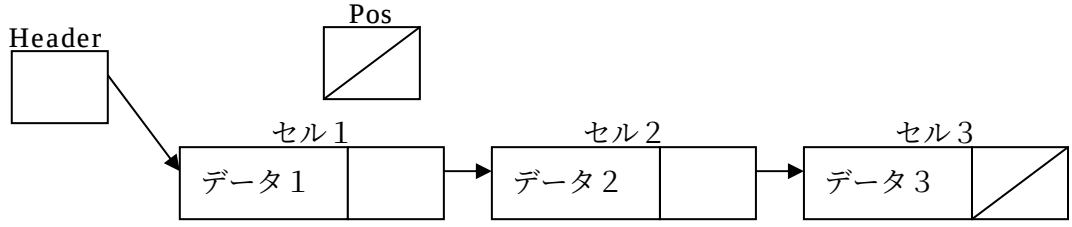
4．Pos は **null** ではないので、セル２のデータを文字列型変数 Data に連結する。

5．参照 Pos を、セル２の参照 Next が指しているセル、つまり、セル３に向ける



6．Pos は **null** ではないので、セル３のデータを文字列型変数 Data に連結する。

7．参照 Pos を、セル３の参照 Next が指しているアドレス、つまり **null** に向ける。



8．Pos は **null** なので、ここで処理を終了し、文字列型変数 Data（の内容）を表示する。

【応用課題 9-A】

上で作成したプログラムに、［新規クラス］作成により、次のような連結リストのクラス LinkList を作成して下さい。

＜クラス LinkList の定義＞

```
class LinkList {
  private Cell Header,Previous;
  public LinkList(String Shimei,int Ten) {
    Cell Temp=new Cell(Shimei,Ten);
    Header=Temp;  //ヘッダが最初のセルを指すようにする
    Temp.Next=null; //ひとまず終端にしておく
    Previous=Temp;  //（次の為に）一つ前のセルを指す参照を用意
  }
  public void AddList(String Shimei,int Ten) {
    Cell Temp=new Cell(Shimei,Ten);
    Previous.Next=Temp; //一つ前のセルが今のセルを指すようにする
    Temp.Next=null; //ひとまず終端にしておく
    Previous=Temp; //（次の為に）一つ前のセルを指す参照を用意
  }
  public String getList() {
    String Data="";
    Cell Pos; //操作対象となるセルを指す参照の用意
    Pos=Header; //セルを指す参照をリストの先頭に向ける
    while (Pos!=null) {  //セルの終端でない限り繰り返す
      Data=Data+Pos.getName()+" "+Pos.getTokuten()+"¥n";
      Pos=Pos.Next; //参照を次のセルに向ける
    }
    return Data;
  }
```

この定義から分かるように、連結リストの生成は LinkList クラスのコンストラクタで行い、さらにセルの追加は AddList メソッドにより行えるようにしています。これを用いれば、［作成］ボタンのプログラムは次のように簡便に記述できます。

＜［作成］ボタン＞

```
  LinkList List1; //LinkList クラスのオブジェクトの宣言

  void jButtonCreate_actionPerformed(ActionEvent e) {
    List1=new LinkList("花形　満",55);   //セル１（最初のセル）を作る
    List1.AddList("轟　次郎",63);  //セル２を追加する
    List1.AddList("金田正太郎",39);  //セル３を追加する
  }
```

これをみて分かるように、いったん上の LinkList クラスが定義されれば、それを利用して

連結リストの生成あるいはセルの追加を行う処理は極めて容易に記述できます。セルの参照を次のセルに向けて・・・などという処理内容の詳細を（LinkList クラスを利用するプログラマは）気にする必要はありません。実際、Cell クラスは上の［作成］ボタンのプログラムには現れなくなりました。これが LinkList クラスを定義したことのメリットです。

さて、それでは［表示］ボタンのプログラムはどのように記述できるでしょうか？以下の空欄を埋めてプログラムを完成させて下さい。上の LinkList クラスの定義をよく読めば分かるはずです。

＜［表示］ボタン＞

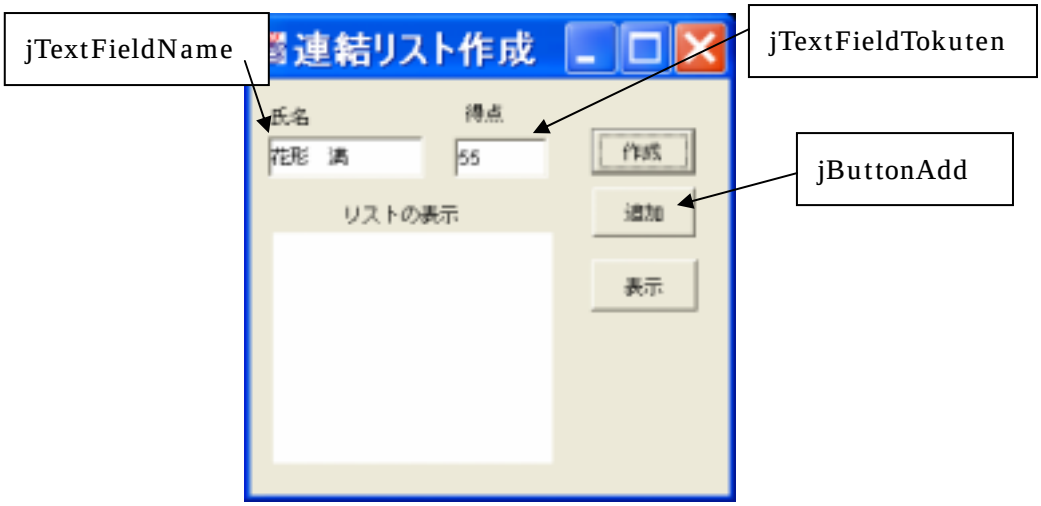
```
void jButtonDisplay_actionPerformed(ActionEvent e) {
  jTextArea1.setText(List1.          );
}
```

作成したらプログラムを実行し、動作を確認して下さい。

【応用課題 9-B】

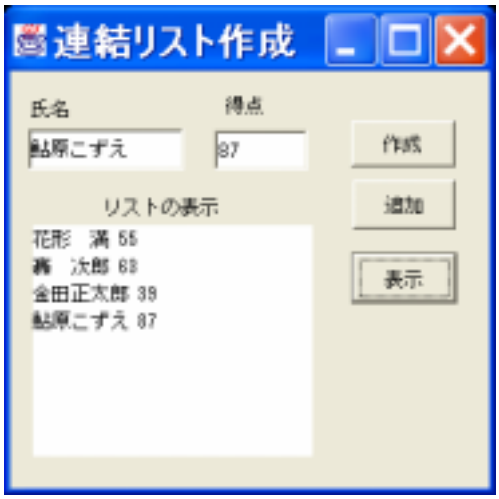
上で作成したプログラムを利用して、今度は一つずつデータを（いくつでも）追加できる様に改良しましょう。連結リストの後ろに順にセルを加えて行くのです。

作成するプログラムの動作内容は以下の通りです。プログラムを起動し、まず氏名欄と得点欄にデータを入力し［作成］ボタンをクリックすると連結リストが新規に作成されます。



※【応用課題 9-A】のプログラムに、上のように氏名、得点欄および［追加］ボタンを追加して下さい。

続けて、追加したいデータを入力後、順次［追加］ボタンをクリックします。これにより、連結リストに追加されます。



幾つかデータを追加した後、［表示］ボタンをクリックすると、それまでに追加したデータがすべて表示されます。

このプログラムの場合、［作成］ボタンのプログラムは次のように修正されます。基本的には、連結リストのオブジェクトを生成すること（のみ）が処理内容です

＜［作成］ボタン＞

```
LinkList List1; //LinkList クラスのオブジェクトの宣言

void jButtonCreate_actionPerformed(ActionEvent e) {
  jTextArea1.setText(""); //表示欄を消去する（白紙に戻す）
  String Name=jTextFieldName.getText();
  int Tokuten=Integer.parseInt(jTextFieldTokuten.getText());
  List1=new LinkList(Name,Tokuten); //連結リストの生成
}
```

次に［追加］ボタンのプログラムは次のようになります。空欄を埋めてプログラムを完成させて下さい。セルを追加するメソッドを呼び出せばよいはずですね。

＜［追加］ボタン＞

```
void jButtonAdd_actionPerformed(ActionEvent e) {
  String Name=jTextFieldName.getText();
  int Tokuten=Integer.parseInt(jTextFieldTokuten.getText());
  List1.                                    ;
}
```

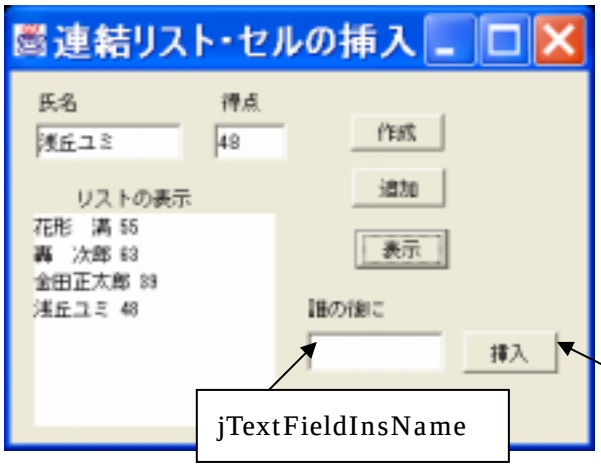
［表示］ボタンのプログラムは【応用課題 9-A】と同じです。変更する必要はありません。

※ 作成したら、実行してみてください。これで、予め要素数を決めておかなくても、いくらでもデータを追加できるプログラムができました。

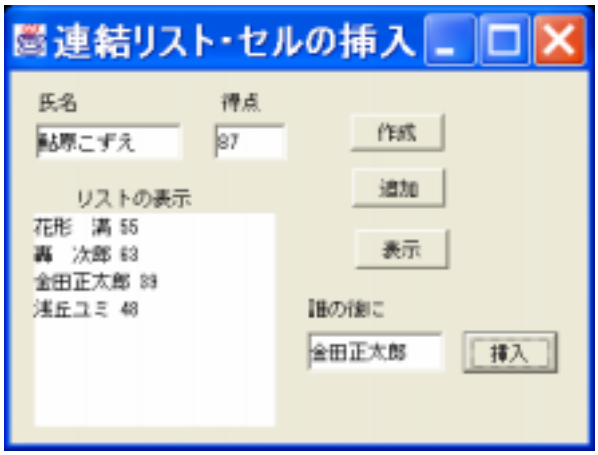
９－４　セルの挿入

【応用課題 9-C】

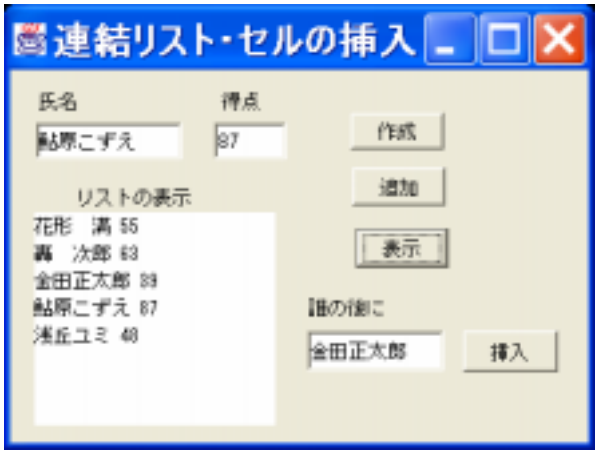
9-2 節で説明したとおり、連結リストの場合、指定したセルの後ろに新たなセルを挿入することは容易に出来ます。そこで、【応用課題 9-B】のプログラムに新たに挿入機能を付け加えましょう。作成するプログラムの動作内容は次の通りです。



前節同様、［作成］および［追加］ボタンを利用してデータを追加します。適当に入力したら［表示］ボタンで全データを表示させます。



次に、「金田正太郎」の後に、「鮎原こずえ」のデータを挿入するものとします。
左の様に入力後、［挿入］ボタンをクリックすると、データが挿入されます。



［挿入］ボタンクリック後、［表示］ボタンをクリックすると、該当箇所にデータが挿入されていることが確認できます。

この挿入処理を行うには、

① 指定した氏名を有するセルを見つける

② そのセルの後に新しいセルを挿入する

という 2 つの処理が必要になります。そこで、クラス LinkList に、これらの処理を行うメソッド InsList(InsName,Name,Tokuten)を次のように追加します（点線枠内)。①および②は上のそれぞれの処理に対応する処理部分です。全体の処理内容は、InsName で指定された氏名のセルの後に、{Name、Tokuten} で指定された（データを有する）セルを挿入するというものです。

```
class LinkList {
  private Cell Header,Previous;
  ・・・
  ・・・  前節までと同様
  public void InsList(String InsName,String Shimei,int Ten) {
    Cell Pos; //挿入対象となるセルを指す参照の用意
    Pos=Header; //セルを指す参照をリストの先頭に向ける
    while ( Pos!=null ) {  //セルの終端まで探索を繰り返す
      if(Pos.getName().equals(InsName)) {
        break;  //指定した氏名を有するセルを見つけたらループから出る
      }
      else {
        Pos=Pos.Next; // 参照を次のセルに向ける
      }
    }                                                          ①
    Cell Temp=new Cell(Shimei,Ten); //挿入セルの生成
    [                                ]
    [                                ]                         ②
}
}
```

②の部分が「参照 Pos が指すセルの後方にセルを挿入する」という処理になるためには、空欄にそれぞれどの式を入れればよいですか？以下の選択肢から選んで下さい。

```
ア  Temp.Next=Pos.Next;
イ  Pos.Next=Temp.Next;
ウ  Pos.Next=Temp;
```

この新たな LinkList クラスのメソッドを用いれば、［挿入］ボタンのプログラムは次のように簡易に記述できます。

＜［挿入］ボタン＞

```
void jButtonIns_actionPerformed(ActionEvent e) {
  String InsName=jTextFieldInsName.getText();
  String Name=jTextFieldName.getText();
  int Tokuten=Integer.parseInt(jTextFieldTokuten.getText());
  List1.InsList(InsName,Name,Tokuten);
}
```

プログラムを作成したら実行し、動作を確認して下さい。

【応用課題 9-D】

上のプログラムは、「誰の後に」欄に指定した氏名がリスト内になかった場合は、エラーになります。これを確認して下さい。

これは、該当する氏名を有するセルがない場合、Pos の値が **null** になった状態で前ページの処理②に進むからです。参照値が null の場合、（指す対象となるオブジェクトが存在しないので）Pos.Next などのようにメソッドを呼び出すことはできません。したがってその段階でエラーになります。

そこで、該当者がいない場合は、最後のセルの後に追加するという様に決めましょう。実は、メソッド LinkList において while ループの継続条件を変更するだけでこの処理を実現できます。空欄を埋めてプログラムを完成させて下さい。

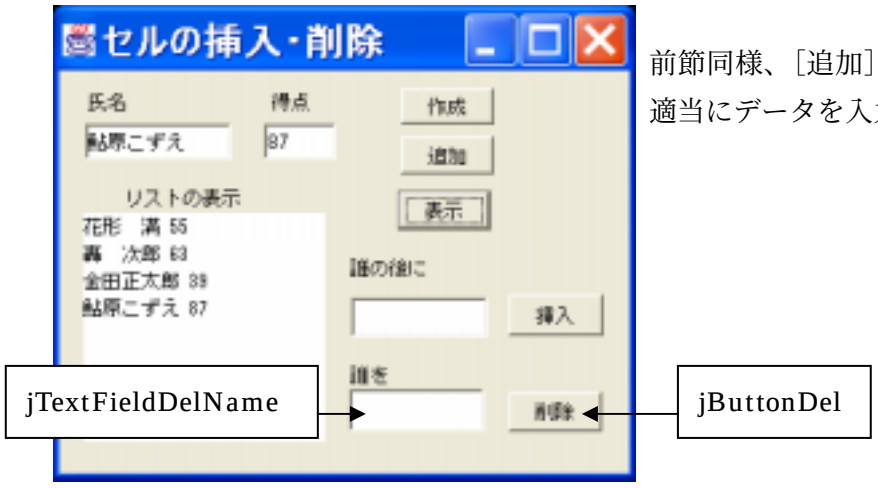
```
public void InsList(String InsName,String Shimei,int Ten) {
  Cell Pos; //探索対象となるセルを指す参照の用意
  Pos=Header; //セルを指す参照をリストの先頭に向ける
  while ( [                ] !=null ) {

    ・・・以下前ページと同じ
}
```

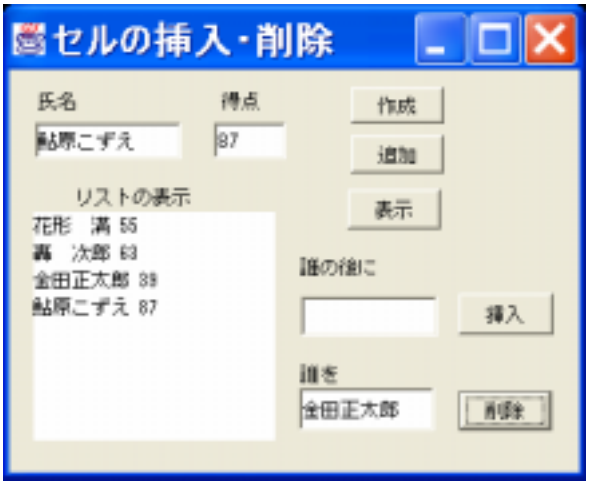
※　これが分かったら、ここまでの内容を良く理解できている、ということになります。

９－５　セルの削除

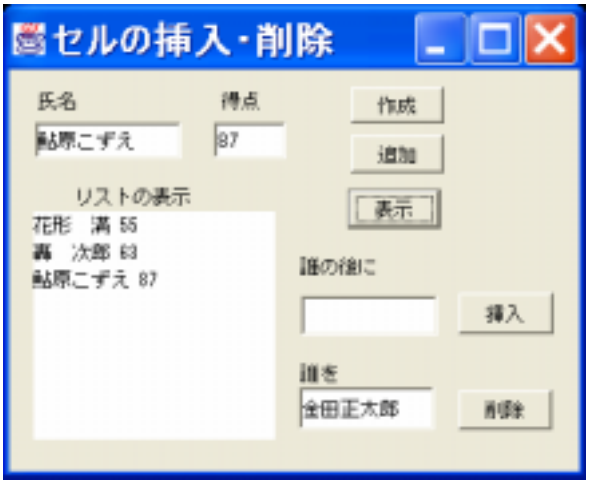
最後に、前節のプログラムにセルを削除する機能を付け加えましょう。ただし、削除の場合、指定したセルの**後ろのセル**を削除する、というやり方はいかにも不自然です。やはり、**指定したセル**を削除する、というようにしたいものです。そこで、次のようなプログラムにしましょう。



前節同様、［追加］ボタンを使って適当にデータを入力します。



入力後、削除したい人の名前を入力し、［削除］ボタンをクリックします。左の例では「金田正太郎」君のデータを削除するようになっています。



［削除］ボタンクリック後、［表示ボタン］をクリックすると、該当データが削除されていることが確認できます。

まず、上のような削除処理を行うメソッドを LinkList クラスに追加しましょう。下の点線枠部分が追加部分です。

```
class LinkList {
  private Cell Header,Previous;
  ・・・
  ・・・  前節までと同様
  public void DelList(String DelName) {
    Cell Pos;     //削除対象となるセルを指す参照の用意
    Cell PrePos; //削除対象の一つ前のセルを指す参照の用意
    Pos=Header; //セルを指す参照をリストの先頭に向ける
    PrePos=Header; //上と同様
    while ( Pos!=null ) {
      if(Pos.getName().equals(DelName)) {
        break;
      }
      else {
        PrePos=Pos;    //一つ前のセルを指す参照を用意する
        Pos=Pos.Next; // 参照を次のセルに向ける
      }
    }
    //Pos が指しているセルをリストから外す
    if(Pos!=null) {  //削除するセルが見つかった場合のみ削除する
      if(Pos==Header) { //削除するセルが先頭の場合の処理
        Header=Pos.Next;
      }
      else {
        PrePos.Next=Pos.Next;
      }
    }
  }
}
```

このメソッドを用いると、［削除］ボタンのプログラムは次のようになります。

```
   void jButtonDel_actionPerformed(ActionEvent e) {
     String DelName=jTextFieldDelName.getText();
     List1.DelList(DelName);
   }
```

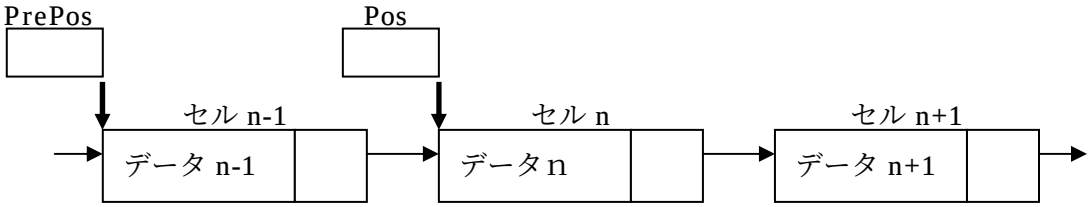
とりあえず実行して動作を確認して下さい。

このプログラムでは、指定したセルのひとつ前のセルを指すポインタ PrePos を用意しており、これが、Pos を追っかける形でくっついて行きます。確認のために削除の過程を以下に簡単に示しておきましょう。

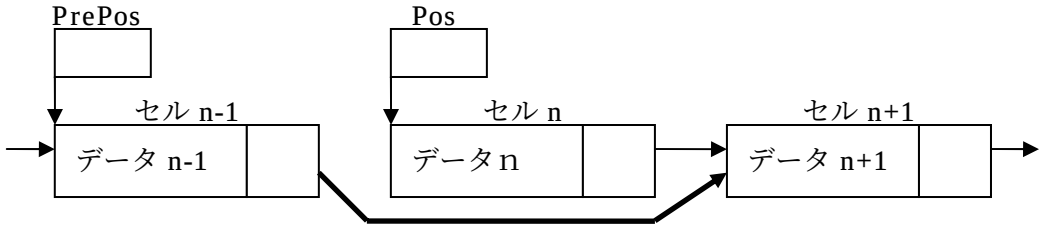
＜セルの削除＞

今、セルｎを削除する場合を考えます。

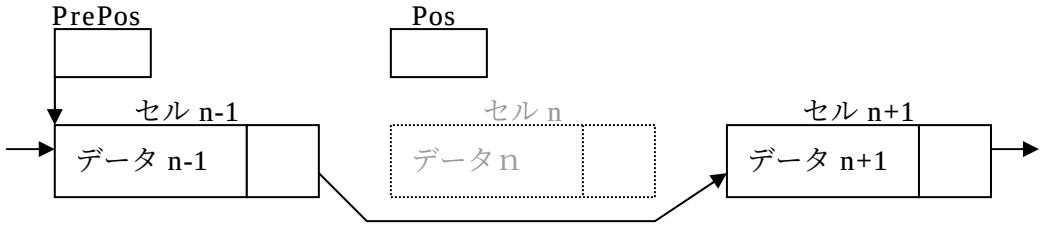
1　削除したいセル（セルｎ）および、その一つ前のセルを指す参照を用意する。



2　PrePos が指すセルの Next（参照）が、Pos が指すセルの Next（参照）と同じセルを指すようにする。



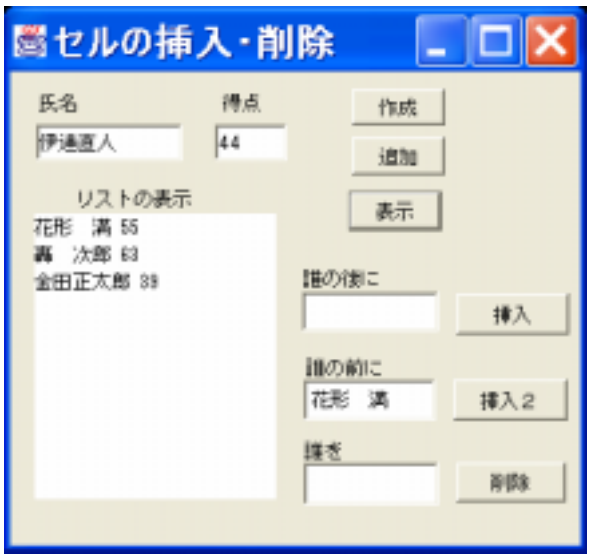
3　これでセルｎはリストから外されました。つまり削除されました。



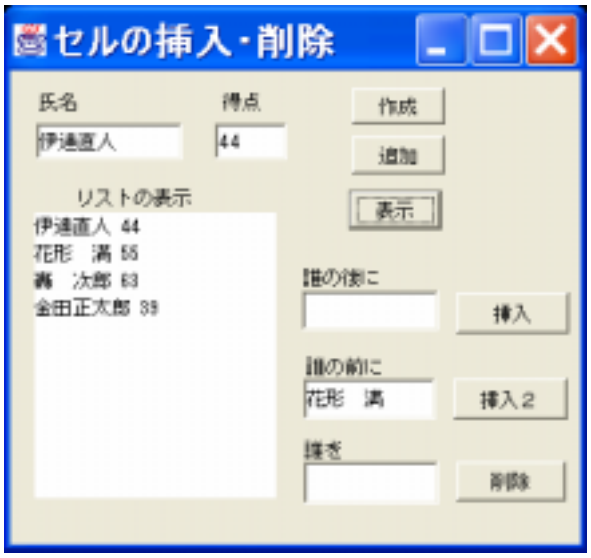
削除完了！

【応用課題 9-E】

上の手法を用いて、下の様に、氏名で指定した人のひとつ前にデータを挿入するボタンを付け加えてください。



適当にデータを入力後、氏名欄と、得点欄に挿入したい人のデータを入力します。
その後「誰の前に」欄に氏名を入力し、［挿入２］ボタンをクリックします。すると、該当者のデータが挿入されます。



［表示］ボタンをクリックすると、指定した人の前に該当者のデータが入力されています。
　もし「誰の前に」欄に指定した氏名のセルがリスト内になかった場合は何も処理しません。

＜ヒント＞

＊　挿入位置を探す部分は、【応用課題 9-C】の挿入と同じです。
＊　セル Pos が指すセルの前に挿入する際には、ポインタ PrePos が必要になります。
＊　挿入する際には、先頭のセルの前に挿入する場合とそれ以外に場合分けする必要があります。

この処理を行うメソッドを LinkList クラスに追加します。そのメソッドの名前を PreInsList とすると、次ページのようになります。空欄を埋めてこのメソッドを完成させ

て下さい。このメソッドを用いれば、［挿入２］ボタンの処理は［挿入］ボタンと同様に容易に記述できるはずです。

＜LinkList クラスへのメソッドの追加＞

```
public void PreInsList(String InsName,String Shimei,int Ten) {
  Cell Pos;     //挿入対象となるセルを指す参照の用意
  Cell PrePos; //挿入対象の一つ前のセルを指す参照の用意
  Pos=Header;   //セルを指す参照をリストの先頭に向ける
  PrePos=Header; //上と同様
  while ( Pos!=null ) {
    if(Pos.getName().equals(InsName)) {
      break;
    }
    else {
      PrePos=Pos;    //一つ前のセルを指す参照を用意する
      Pos=Pos.Next; // 参照を次のセルに向ける
    }
  }
  Cell Temp=new Cell(Shimei,Ten); //挿入セルの生成
  if(Pos!=null) {      //挿入位置のセルが見つかった場合のみ挿入する
    if(Pos==Header) { //挿入するセルが先頭の場合の処理
      [                                    ]

      [                                    ]

    }
    else {
      [                                    ]

      [                                    ]

    }
  }
}
```

【自由課題】

みなの中には、自分で自由にテーマを決めてプログラムを作りたいという希望を持っている人もいることと思います。そこで、自作のプログラムを評価して欲しいという人は、森田まで提出して下さい（ノート PC を持って来てデモンストレーションを見せて下さい）。内容に応じて加点します（最大１５点まで）。ただし、オリジナルのものに限ります。〆切は 1/13 の演習時間終了時までです。