第９章　補足

－Java プログラムを一から記述してみよう－

【学習内容とねらい】

これまで本テキストで扱ってきたのは主に Windows アプリケーション（Windows 上のボタンクリック等による動作するプログラム）でした。ですから、皆は Eclipse を用いて Windows アプリケーションを作成する方法には習熟したはずです。

皆も経験した通り、Eclipse（およびそこにプラグインされている Jigloo GUI Builder）では、フレームの設計やイベント処理に関する多くの定型処理を自動的に記述してくれるので非常に便利です。しかしその反面、自動生成された NewJFrame.java のソースを眺めてみても、意味がよく分からない部分が多々あり不安あるいは不満を感じた人もいたことと思います。確かにそれら詳細が分からなくてもプログラムを作成できるところが Eclipse のメリットなのですが、より本格的に Java プログラミングを学習したい人にとっては、それら、言わばこれまで Eclipse に任せて来た部分を自分で理解しておく必要があります。そのためには、Eclipse のコード生成機能を用いずに一から自分（だけ）でプログラムを作成してみるのが一番です。

そこで、本章では、簡単な Windows アプリケーションを自分で最初から記述してみることにしましょう。本章の説明に沿ってプログラムを作成して行くと、これまで作成してきた Java アプリケーションプログラムの全体像が見えてくる筈です。そうすることで、Eclipse の便利さも改めて理解できる様になり、したがってそれをより使いこなせるようになると思います。

＜第9章の構成＞

9-1 フレームのないプログラムの作成

まず、手始めにフレームのないプログラムを考えましょう。それは 1-4 節で学習したような、コンソール画面（コマンドプロンプト画面）に結果を表示するようなプログラムです。1-4 節では適当なエディタを用いてプログラムを記述し、その後でコンソール画面からコマンドを入力してプログラムを実行させました。これが Eclipse などの統合開発環境を用いない場合の、Java プログラムの作成・実行のやり方です。では、Eclipse を用いてもこのような（フレームのない）プログラムを作成・実行することはできるでしょうか？もちろん可能です。Java プログラムを一から記述する学習としてちょうど良い肩慣らしになるので、以下にその作成方法を学習しましょう。作成するのは画面に「Hello Java!」と表示するプログラムです。

【例題 9-1-1 「Hello Java!」プログラム】

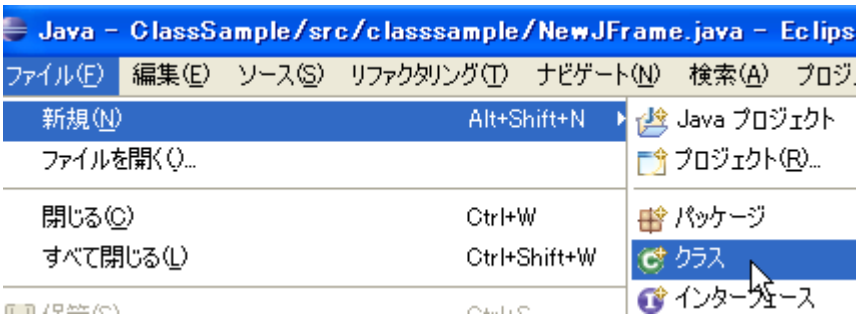
次の手順にしたがってプログラムを作成して下さい。

① Java プロジェクトの新規作成

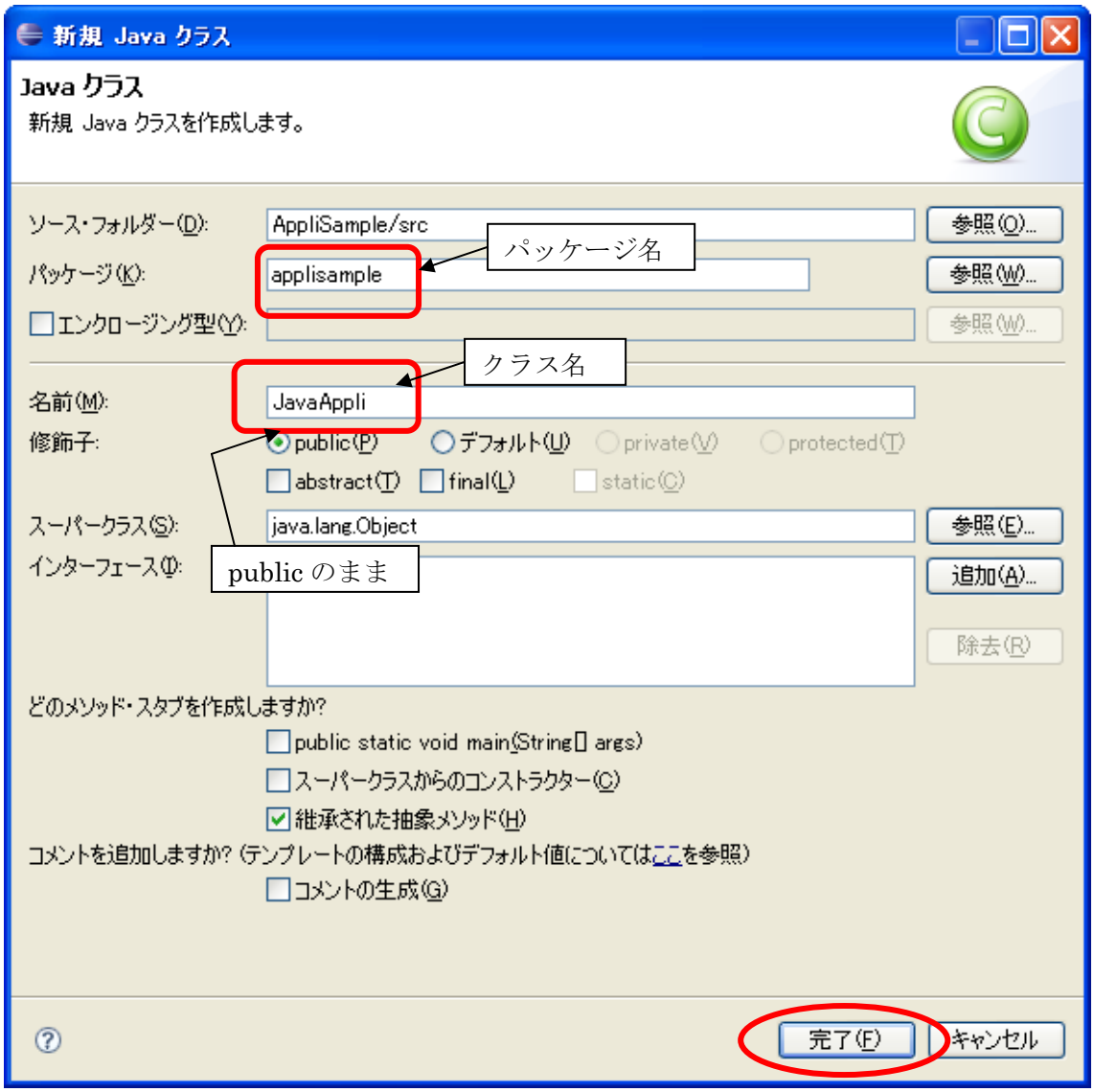
Eclipse で Java プログラムを作る際には（フレームを用いない場合でも）必ず、これまで同様プロジェクトの新規作成から始めます。これは、Eclipse 内でプログラムを実行させるために必要なのです。ここでは、プロジェクト名を「AppliSample」として新規作成して下さい。

② クラスの新規作成

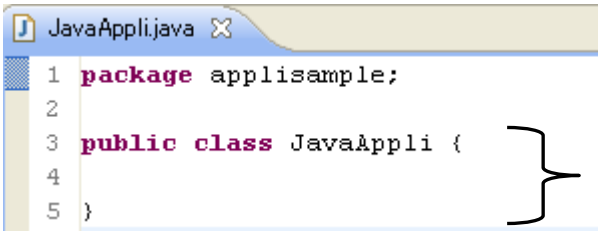
その後、今作成したプログジェクト内にクラスを新規作成します。7 章で学習した様に、ワークベンチの「ファイル」メニューから「新規」→「クラス」を選択して下さい。



そして、クラス設計ウィザードで次ページのように設定します。図から分かる通り、クラス名を「JavaAppli」、パッケージ名を「applisample」と指定します（これらの名前は何でも良いのですが以下ではこのように指定したものとして説明します）。



設定後［完了］ボタンをクリックすると、次の編集画面が現れます。

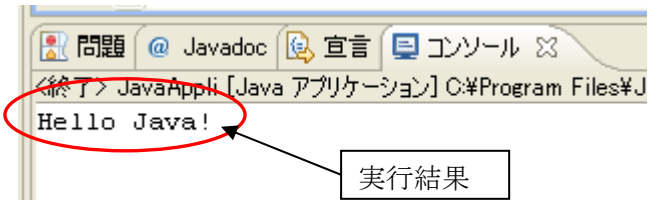


③　プログラムの作成・実行

ここで、「クラス定義記述部分」を次のように記述して下さい。

```
public class JavaAppli {
    public static void main(String[] args) {
          System.out.println("Hello Java!");
    }
}
```

これは、1-4 節（p.25）で記述したものと（クラス名を除いて）同じです。このプログラムを通常通り実行すると、次のように結果がワークベンチ下方のビューにある「コンソール」画面に現れます。



このように、コンソール画面はコマンドプロンプト画面と同様の働きをします。

以上のように、（Windows アプリケーションに限らず）どのような Java プログラムでも Eclipse を用いて作成・実行することができます。

【例題 9-1-2　データの入力】

さて、もう一度 main メソッドを眺めてみましょう。

```
public static void main(String[] args) {
  System.out.println("Hello Java!");
}
```

これを見ると、main メソッドは引数(String[] args)を持っています。これは何のために必要なのでしょうか？恐らく疑問に思った人も多いでしょう。実はこれは実行時にデータを入力する際に必要となる変数なのです。

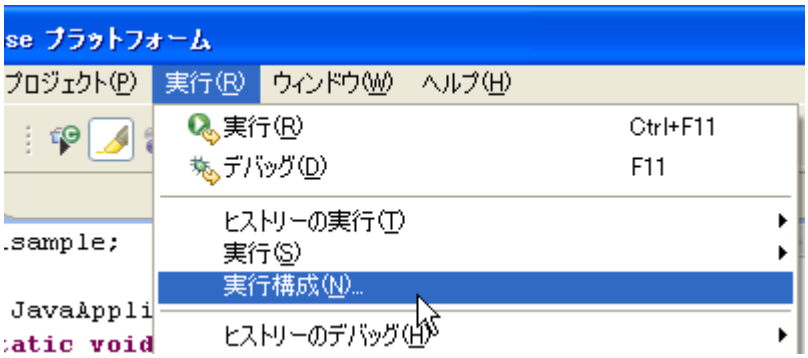
まず、引数「args」は文字列型の配列であることが分かるでしょう（配列は 4-12 節で学習しました）。ですからargsはargs[0]、args[1]、･･･の様な形で複数の値を持ち得ます。この引数の意味（役割）を理解するには、実際に入力データを使用した実例を確認するのが早道です。そこで、以下のその例を学習しましょう。

まず、main メソッドに以下の枠線部を追加して下さい。

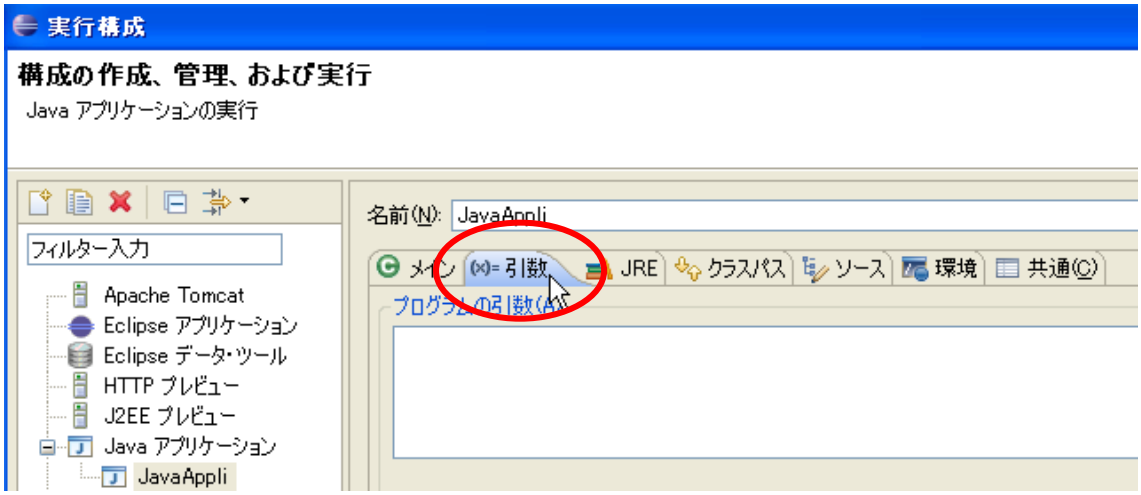
```
public class JavaAppli {
  public static void main(String[] args) {
    System.out.println("Hello Java!");

    int a=Integer.parseInt(args[0]);
    int b=Integer.parseInt(args[1]);
    System.out.println(a+b);

  }
}
```

枠線部は、『0 番目と 1 番目の引数を整数 a,b として受け取りその合計を画面に表示する』という処理になっています。

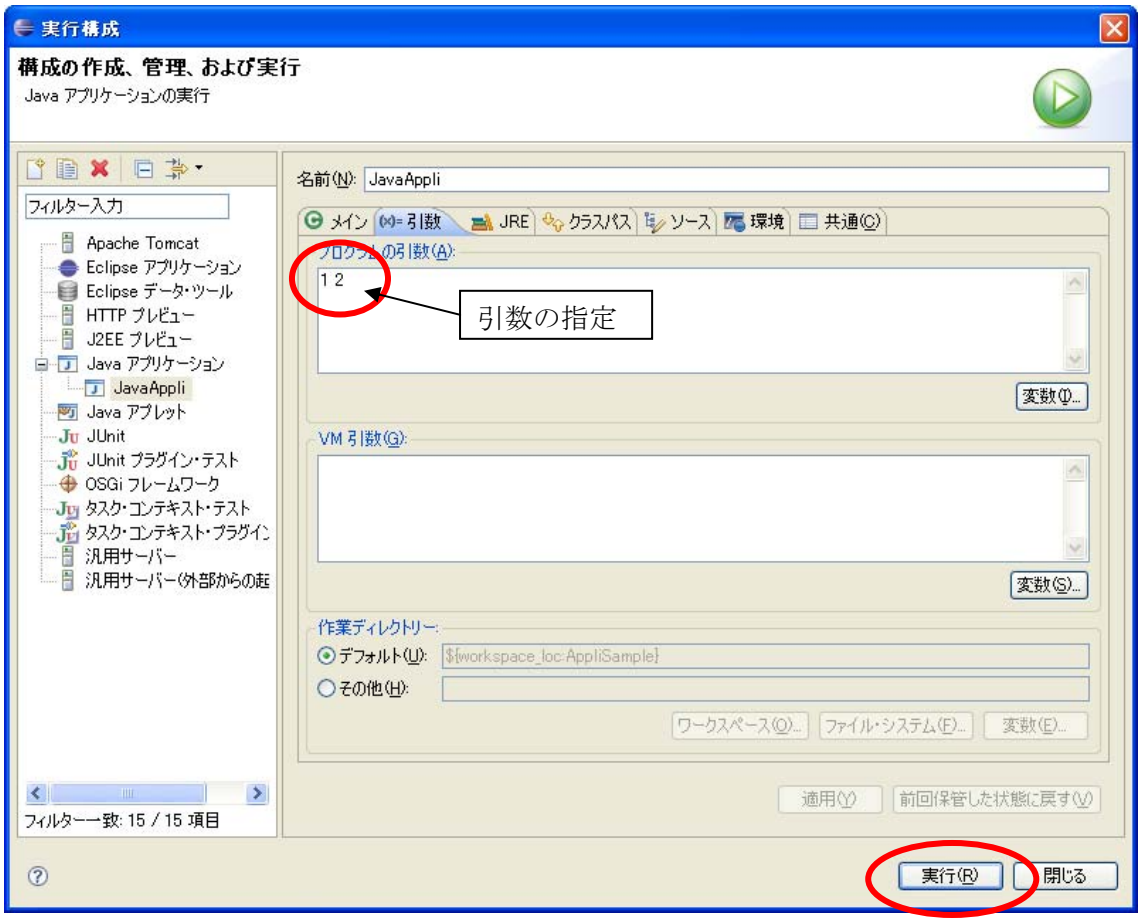
さて、Eclipse を用いてプログラム実行時に入力データを指定するためには、上のようにプログラム記述後、「実行」→「実行構成」を選択します。



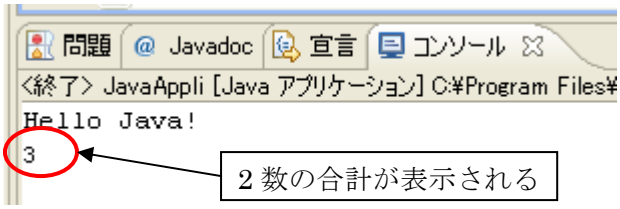
その後、現れた「実行構成」画面で「引数」タブを選択します。



すると、上のような画面が現れるので、次ページのように「プログラムの引数」欄に適当な２つの整数（下の例では１と２）を空白で区切って入力して下さい。なお、今の場合、整数入力なので入力は全て半角で行ってください。



入力後、画面下の［実行］ボタンをクリックすると、下のように、2 数の合計が結果として表示されます。

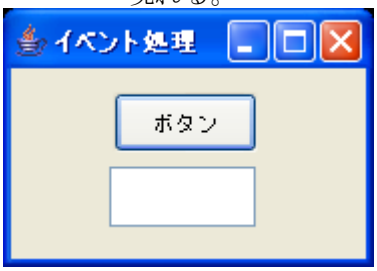


これで、main メソッドの引数 args の意味（役割）が分かったでしょう。

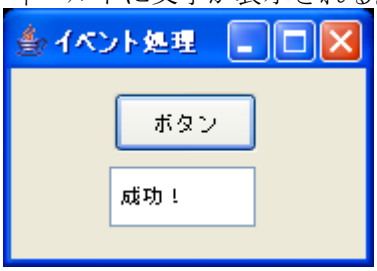
9-2 フレームの生成・表示

本節以降では、次のようなプログラムを作成する事にします。

プログラムを起動すると次の画面が現れる。



ボタンをクリックすると、テキストフィールドに文字が表示される。



本節ではまず、フレームを生成してそれを表示させる（だけの）プログラムを作成しましょう。もちろん、GUI Editor のフレーム設計機能を用いずに自力で作成します。

【例題 9-2-1 フレームの生成・表示】

まず、前節で作成したプログラムを開いた状態にしておいて下さい。
ここで、以下のようにプログラムを修正して下さい。具体的には、波線部を追加し、main メソッドを枠線部のように修正しています。

```
package applisample;

import javax.swing.*; //フレームを用いるために必要      ①

public class JavaAppli {
  public static void main(String[] args) {
    JFrame frame=new JFrame(); //フレームの生成                ②
    frame.setTitle("イベント処理"); //フレームタイトルの設定      ③
    frame.setBounds(100,100,200,150); //フレームサイズの設定    ④
    frame.setVisible(true); //フレームの視覚化                 ⑤
  }
}
```

＜プログラムの解説＞

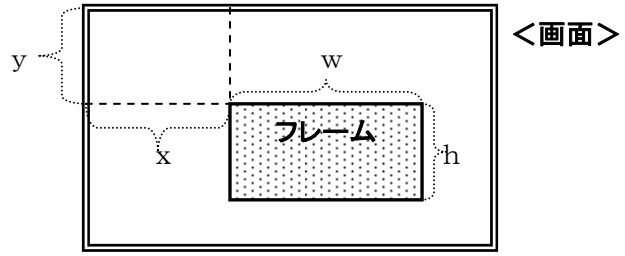
① フレームクラスは swing というパッケージに入っています。この中には、ボタンやテキストフィールドなどの各コンポーネントも含まれています。ですから、それらコンポーネントを用いるときには、冒頭でこのように swing パッケージを指定します。

② フレームは JFrame クラスというクラスで定義されています。ここでは、JFrame クラスのオブジェクトを frame という名前で生成しています。オブジェクトの生成につい

て不明な点があれば第 7 章を読み返して下さい。

③ JFrame クラスには、`setTitle()`メソッドが定義されており、フレームのタイトルを引数として指定します。

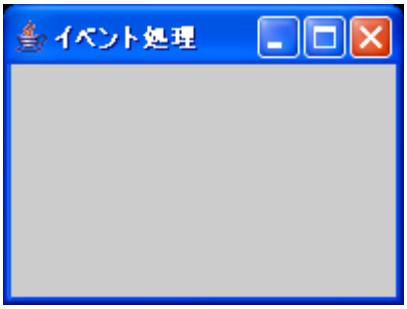
④ `setBounds(x,y,w,h)`メソッドは、画面上の座標(x,y)の位置に、幅 w、高さ h の大きさでフレームを配置するメソッドです。



⑤ `setVisible(true)`メソッドにより、フレームを画面に表示できます。引数のデフォルト（既定値）は false に設定されているので、この文がなければプログラムを実行してもフレームは表示されません。少し不自然に思うかもしれませんが、複数のフレームを扱う場合などを想定するとフレームを表示させるタイミングはプログラマが決めた方が、一般には便利なのです。

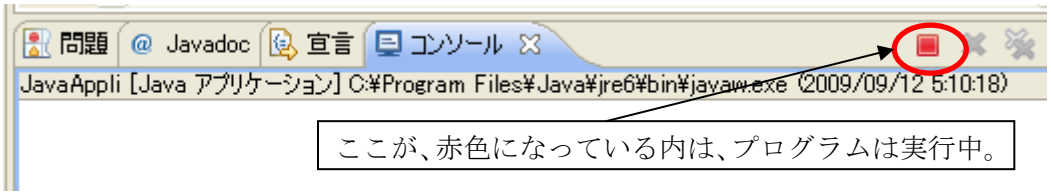
これらは、これまでは GUI Editor が自動生成していた部分なので、初めて目にする部分もあると思いますが、いずれも意味は極めて単純なので理解に問題はないでしょう。

さて、作成したらプログラムを実行しましょう。実行すると、次のようなフレームが現れるはずです。



これで、実行時にウィンドウ（フレーム）を表示させるプログラムが出来ました。ひとまず、このフレームを閉じましょう。

ところが、「コンソール」ビューを見ると、次のようにまだプログラムが終了していないことが分かります。



実は、このままでは、フレームの右上隅の×をクリックしても、非表示になるだけでプログラムは終了しないのです。ひとまず、上の赤ボタンをクリックしてプログラムを終了さ

せて下さい。

×をクリックしたときにプログラムを終了させるためには、「×をクリックしてフレームを閉じる」というイベントが発生したときに、プログラムも終了する様にプログラムを記述する必要があります。次の例題プログラムに進んで下さい。

【例題 9-2-2 プログラムの終了処理の追加】

上の例題プログラムに次の枠線部を追加して下さい。

```
package applisample;

import javax.swing.*; //フレームを用いるために必要

public class JavaAppli {
  public static void main(String[] args) {
    JFrame frame=new JFrame(); //フレームの生成
    frame.setTitle("イベント処理"); //フレームタイトルの設定
    frame.setBounds(100,100,200,150); //フレームサイズの設定
    frame.setVisible(true); //フレームの視覚化
    frame.setDefaultCloseOperation(
                          WindowConstants.DISPOSE_ON_CLOSE);
  }
}
```

＜プログラムの解説＞

- setDefaultCloseOperation()は、フレームクラスに定義されているメソッドで、フレームが閉じられた際に行う処理を指定します。
- 処理は、(　)内の引数に 0～3 のいずれかの番号（整数）を指定することで行われます。デフォルトでは、0 が指定されており、これは、フレームが閉じられても何もしない事を意味します。
- 実は、"WindowConstants.DISPOSE_ON_CLOSE"は所定の定数を意味し、具体的には 2 です。2 という処理は、閉じた後フレームをメモリから消去する、ということを意味します。今の場合、フレームが全くなるのでプログラムは終了します。
- ですから、上の枠線部のプログラムは、次のように書いても同等です。

```
frame.setDefaultCloseOperation(2);
```

プログラムを作成したら実行し、フレームを閉じてみて下さい。今度はプログラムも終了するはずです。これを確認して下さい。

10-3 コンポーネントの貼り付け

本節では、前節で作成したフレームにボタンとテキストフィールドを貼り付けましょう。【例題 9-2-2】で作成したプログラムを開いておいて下さい。

【例題 9-3-1 コンポーネントの貼り付け】

【例題 9-2-2】のプログラムに以下の波線部および枠線部を追加して下さい。

```
import javax.swing.*;  //フレームを用いるために必要
import java.awt.*;  //Container クラスを用いるために必要     ①

public class JavaAppli {
  public static void main(String[] args) {
    JFrame frame=new JFrame(); //フレームの生成
    frame.setTitle("イベント処理"); //フレームタイトルの設定
    frame.setBounds(100,100,200,150); //フレームサイズの設定
    //コンポーネントの生成          ②
    JButton jBtn=new JButton(); //ボタンの生成
    jBtn.setBounds(50,20,100,30); //ボタンサイズの設定
    jBtn.setText("ボタン"); //ボタンの表示テキストの設定
    JTextField jTxt=new JTextField(); //テキストフィールドの生成
    jTxt.setBounds(50,70,100,30); //テキストフィールドサイズの設定
    //コンポーネントを貼り付ける
    Container Cont=frame.getContentPane();     ③
    Cont.setLayout(null); //null レイアウトの指定   ④
    Cont.add(jBtn); //ボタンの貼り付け
                                                    ⑤
    Cont.add(jTxt); //テキストフィールドの貼り付け
    frame.setVisible(true); //フレームの視覚化
    frame.setDefaultCloseOperation(
                          WindowConstants.DISPOSE_ON_CLOSE);
  }
}
```

＜プログラムの解説＞

① コンポーネントは直接フレームの上に貼り付けられません。コンポーネントを貼り付けるためには、貼り付けが可能な機能（**コンテナ**機能）を持ったクラスのオブジェクトである必要があります。代表的なものがそのものずばりの Container（コンテナ）クラ

スです。この Container クラスは java.awt パッケージに含まれているのでここでそれを指定しています。なお、JPanel クラスもコンテナ機能を持っています。

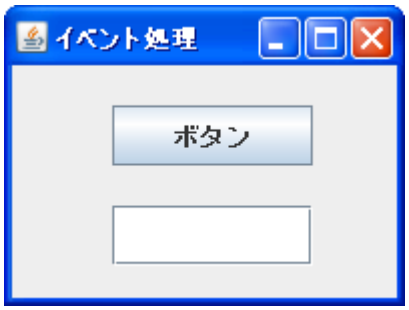
② ボタンコンポーネントおよびテキストフィールドコンポーネントのクラスはそれぞれ JButton および JTextField です。それが分かればこの「コンポーネントの生成」部分は理解できるでしょう。なお、`setBounds(x,y,w,h)`メソッドの意味は【例題 9-2-1】（p.242）で説明した通りですが、今の場合、これらコンポーネントはフレームの上に配置されることになるので、配置位置（`x,y`）は、フレームの左上隅を原点（`0,0`）としたときの座標となります。

③ 右辺の `frame.getContentPane()`により、フレームコンポーネント `frame` に付随した**コンテナ**を取得することができます。そしてそれを①で説明した Container クラスのオブジェクト `Cont` に代入することで、以後、`Cont` を `frame` のコンテナとして用いる事が出来ます。`Cont` は（`Container` クラスのオブジェクトなので）その上にコンポーネントを貼り付ける事ができます（⑤参照）。そうして、`Cont` にコンポーネントを貼り付ければ、結果的にフレームの上に貼り付けられることになります。

④ ここで、レイアウトを null に指定しています。これは、「Absolute Layout」のことです。これまでは、GUI Editor 上でレイアウトを指定していましたが、その際、Eclipse がこのような文を自動的に生成してくれていたのです。

⑤ コンテナ Cont にコンポーネントを貼り付けるためには、`add("コンポーネント")`メソッドを用います。

プログラムを作成して実行すると、次の画面が現れます。ただ、まだボタンクリック時のイベント処理は記述していません。それは次節で行います。



さて、次節に進む前に、少しプログラムを整理しておきましょう。通常、`main()`メソッドには細かい具体的な処理はあまり記述しないようにします。そして、個別の処理はメソッドに定義しそれらを `main()`メソッドで呼び出すという形式を採ります。その方が処理内容の全体を見渡せ、見通しが良くなるからです。

そこで、ここでも、（一歩進んだプログラミングを目指して）処理の詳細の記述を新たなメソッドに移すようにしましょう。次のようなメソッド `initGUI()`をクラス内に定義して下さい。

```
void initGUI() {
    JFrame frame=new JFrame(); //フレームの生成
    frame.setTitle("イベント処理"); //フレームタイトルの設定
    frame.setBounds(100,100,200,150); //フレームサイズの設定

    //コンポーネントの生成
    JButton jBtn=new JButton(); //ボタンの生成
    jBtn.setBounds(50,20,100,30); //ボタンサイズの設定
    jBtn.setText("ボタン"); //ボタンの表示テキストの設定
    JTextField jTxt=new JTextField(); //テキストフィールドの生成
    jTxt.setBounds(50,70,100,30); //テキストフィールドサイズの設定
    //コンポーネントを貼り付ける
    Container Cont=frame.getContentPane();
    Cont.setLayout(null); //null レイアウトの指定
    Cont.add(jBtn); //ボタンの貼り付け
    Cont.add(jTxt); //テキストフィールドの貼り付け

    frame.setVisible(true); //フレームの視覚化
    frame.setDefaultCloseOperation(
                          WindowConstants.DISPOSE_ON_CLOSE);
}
```

これは、p.244 に示した main()メソッドの中身をそっくりコピーしただけです。

このメソッドを用いると、main()メソッドは次のように書き換えることができます。

```
public static void main(String[] args) {
   JavaAppli jAppli1=new JavaAppli();  //オブジェクトの生成
   jAppli1.initGUI();     //フレーム作成処理を行うメソッドの呼び出し
}
```

つまり、main()メソッドで行うのは、

①（今作成しているクラスの）オブジェクトを生成し、

② フレーム作成等、必要な処理を行うメソッドを呼び出す。

という 2 点になります。これが、Web アプリケーションを作成する場合の典型的な main()メソッドの記述内容です。

このように記述しておけば、フレームへのコンポーネントの配置やイベント処理の内容等に変更があっても、それは所定のメソッド内の記述を変更すれば良く、main()メソッドは変更する必要はなくなります（実際、次節以降プログラムを改良して行きますが、main()メソッドはいじる必要はなく、このままです）。

9-4 イベント処理の追加

それでは、最後に、［ボタン］をクリックした時に、テキストフィールドに文字が表示されるようにイベント処理を追加しましょう。前節で作成したプログラムを開いておいて下さい。

【例題 9-4-1 ボタンイベントの追加】

次の手順でイベント処理を追加します。

Ⅰ import 文の追加

【例題 9-3-1】のプログラムに以下の import 文（波線部）を追加して下さい。

```
import javax.swing.*;  //フレームを用いるために必要
import java.awt.*;  //Container クラスを用いるために必要     ①
import java.awt.event.*; //イベント処理記述のために必要

public class JavaAppli {
  public static void main(String[] args) {
      JavaAppli jAppli1=new JavaAppli();  //オブジェクトの生成
      jAppli1.initGUI();     //フレーム作成処理を行うメソッドの呼び出し
  }

   void initGUI() {
      JFrame frame=new JFrame(); //フレームの生成
      ・・・
     frame.setDefaultCloseOperation(
                              WindowConstants.DISPOSE_ON_CLOSE);
   }
}
```

Ⅱ-②で必要になる ActionListener インターフェース（p.250 参照）は、`java.awt.event` というパッケージに含まれています。そのために、このパッケージを指定しておく必要があるのです。

Ⅱ イベントリスナの登録

メソッド `initGUI()`の末尾に下線部を加えて下さい。

```
void initGUI() {
     ・・・
    BtnAction BAct=new BtnAction(jTxt);
    jBtn.addActionListener(BAct);      //イベントリスナの登録  ①
}
```

BtnAction クラスは次ページで定義

そして、JavaAppli クラス内にクラス BtnAction の定義を加えて下さい。

```
public class JavaAppli {
  public static void main(String[] args) {
      . . .
  }
  void initGUI() {
      . . .
  }

  class BtnAction implements ActionListener {     ②
     JTextField tfd;  //テキストフィールドをフィールド変数として宣言 ⑤
     public BtnAction(JTextField jTxt) { //コンストラクタ  ④
           tfd=jTxt; //引数のjTxtをフィールド変数tfdに代入
     }
     public void actionPerformed(ActionEvent evt){    ③
           tfd.setText("成功！");
     }
  }
}
```

＜プログラムの解説＞

① `jBtn.addActionListener("`**イベントリスナ**`")`メソッドという形で、ボタンオブジェクト（`jBtn`）に**イベントリスナ**を登録します。イベントリスナについては p.249 のコラムを参照して下さい。ボタンに関するイベントはクリックしかないので、ボタンクリックに関するイベントリスナと言っても良いです。そのイベントリスナは 1 行上に記述している様に `BtnAction` というクラスのオブジェクト `BAct` であり、コンストラクタの引数として（処理対象となる）テキストフィールドを受け取ります。

② クラス `BtnAction` は `ActionListener` という**インターフェース**を**実装（継承）**して定義されます。インターフェースについては p.250 のコラムを参照して下さい。ここでは、『ボタンクリック・イベント発生時に呼び出されるメソッド（つまり `actionPerformed()`）がその名前だけ定義されているクラス』だと捉えて下さい。そして **`implements`** は**継承(`extends`)**と意味は同じです。**継承**については 7-4 節を参照して下さい。

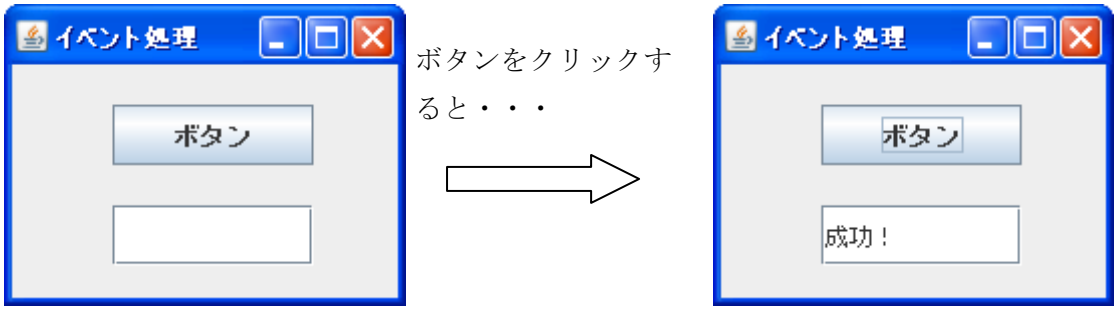
③ ボタンクリック時に呼び出されるメソッドは `actionPerformed ()`というメソッドです。そこで、ここに処理内容、つまり『(フレーム上の）テキストフィールドに「成功！」という文字列を表示させる』と言う処理を記述します。

④ ③の処理を実行するためには、メインクラスから当該テキストフィールド `jTxt` を受け取らなければなりません。そのために、コンストラクタの引数に（処理対象となる）テキストフィールドを受け取れるようにしているのです。そして引数として受け取っ

たテキストフィールド変数 jTxt を、③の処理で使用できるようにフィールド変数 tfd に代入しています。引数は当該メソッド内でのみ有効な**ローカル変数**である点に注意して下さい。一方、フィールド変数はクラス内でアクセス可能なので③のメソッド actionPerformed ()内でも処理できるのです。

⑤ そのために、JTextField クラスの変数をフィールド変数として宣言しておく必要があります。

プログラムを作成したら実行して動作を確認して下さい。これで、GUI Editor を用いずに（自力で）イベント処理を記述する事ができました。他のイベントも同様に記述する事ができます。



コラム　イベント処理について

Java 言語開発グループは、イベント処理が、**イベントソース**、**イベント**そして**イベントリスナ**の 3 者から構成されるものと捉えました。上の例の場合、イベントソース、すなわちイベントの発生元はボタンです。そしてイベントは、「ボタンをクリックする」という事象で、イベントリスナは BtnAction クラスのオブジェクトです。

Java言語では、イベント処理を記述する場合、イベントソースにイベントリスナを登録します。イベントリスナは“聞き役”という意味ですが、言わばイベントの監視役です。上の例で言うと、addActionListener("**イベントリスナ**")という形で（ボタンに）イベントリスナを登録しています。これにより、ボタンにイベント発生を感知する機能が備わり、イベント（今の場合ボタンクリック）が発生した場合、イベントリスナは、自身に定義された所定のメソッド（今の場合はActionPerformed()）を起動する、という仕掛けになっています。

コラム　インターフェースとは

インターフェースとは、名前のみが決まっていて処理内容を定義していないメソッド（群）からなるクラスの事です。処理内容は場合に応じて異なってもメソッドの名前は共有したい、つまりメソッド定義を標準化したい、という考えから導入されました。

ただし、インターフェースに含まれるメソッド群については、それを継承したクラスにおいて、（何もしないと言う処理を含めて）何らかの処理を定義する必要があります（定義しないメソッドがあるとエラーになります）。

なお、**implements** は上の＜プログラムの解説＞でも述べた通り、継承（extends）と意味は同じです。継承元のクラス（スーパークラス）がインターフェースの場合のみ implements となります。

9-5 プログラムの整理

前節まででプログラムは完成しました。しかし、もう少しプログラムを整理しておきましょう。前節で作成したプログラムを開いておいて下さい。

【例題 9-5-1　BtnAction クラスの書き換え】

前節のプログラムでは、BtnAction クラスのコンストラクタで、処理対象であるテキストフィールドコンポーネント `jTxt` を引数として渡していました。しかし、これらコンポーネントをインスタンス変数として宣言しておけば(BtnAction クラスから直接アクセスできるので)、このような手続きは不要になります。そこで、`frame`、`jBtn` そして `jTxt` をインスタンス変数として宣言するように変更しましょう。

次ページのように JavaAppli クラスを修正して下さい。枠線部を追加し、下線部を修正しています。また、BtnAction クラスでは、テキストフィールドコンポーネントの引き渡しが不要になったので削除しています。なお、p.246 で説明した通り、このような変更があっても `main()`メソッドは全くいじる必要がありません。

修正したらきちんと動作することを確認して下さい。もちろん、実行結果は変わりません。確認が済んだら、本章の仕上げとして次の例題に進んで下さい。

```
public class JavaAppli {
   JButton jBtn;
   JFrame frame;
   JTextField jTxt;

   public static void main(String[] args) {
        ・・・（変更無し）
    }

    void initGUI() {
        frame=new JFrame(); //フレームの生成（宣言は外す）
        frame.setTitle("イベント処理"); //フレームタイトルの設定
        frame.setBounds(100,100,200,150); //フレームサイズの設定
        //コンポーネントの生成
        jBtn=new JButton(); //ボタンの生成（宣言は外す）
        jBtn.setBounds(50,20,100,30); //ボタンサイズの設定
        jBtn.setText("ボタン"); //ボタンの表示テキストの設定
        jTxt=new JTextField(); //テキストフィールドの生成（宣言は外す）
        jTxt.setBounds(50,70,100,30); //テキストフィールドサイズの設定
        //コンポーネントを貼り付ける
        Container Cont=frame.getContentPane();
        Cont.setLayout(null); //null レイアウトの指定
        Cont.add(jBtn); //ボタンの貼り付け
        Cont.add(jTxt); //テキストフィールドの貼り付け
        frame.setVisible(true); //フレームの視覚化
       frame.setDefaultCloseOperation(
                              WindowConstants.DISPOSE_ON_CLOSE);
      BtnAction BAct=new BtnAction();
      jBtn.addActionListener(BAct);      //イベントリスナの登録
    }

    class BtnAction implements ActionListener {
      public void actionPerformed(ActionEvent evt){
            jTxt.setText("成功！");
      }
    }
}
```

コンポーネントの宣言をここで行う（インスタンス変数とする）。

コンポーネントをインスタンス変数とするため、メソッド内では、オブジェクトの生成のみ行う。

引数は不要になる。

jTxt はインスタンス変数であるから、直接用いる事ができる。

【例題 9-5-2　イベントリスナ登録の改良】

上の例題で一通りの整理は出来たのですが、本章の最後に、ボタンをクリックした時のイベントリスナの登録の仕方をより拡張性のあるものに改良しましょう。

今の場合ボタンに対するイベントリスナを登録するため、下のように BtnAction という名前のクラスを用意しました。

```
void initGUI() {
    ・・・
   BtnAction BAct=new BtnAction();
   jBtn.addActionListener(BAct);      //イベントリスナの登録
}
class BtnAction implements ActionListener {
   public void actionPerformed(ActionEvent evt){
          jTxt.setText("成功！");
   }
}
```

しかし、よく考えるとイベントリスナは、addActionListener()メソッドの引数として用いるためだけに必要なので、そのためにわざわざクラスを宣言するのは煩わしい気がします。イベント処理の数が増えた場合はなおさらです。

このように、あるクラスが特定の場所のみに用いられる場合、Java 言語ではそれを宣言せずに用いることができるようになっています。具体的には上のプログラムの場合、次のように書き換えることができます。

```
void initGUI() {
    ・・・
   BtnAction BAct=new BtnAction();
   jBtn.addActionListener(new ActionListener {
      public void actionPerformed(ActionEvent evt){
         jTxt.setText("成功！");
      }
    });    //イベントリスナの登録
}
class BtnAction implements ActionListener {
   public void actionPerformed(ActionEvent evt){
          jTxt.setText("成功！");
   }
}
```

この部分は不要になる。

ここに直接書く。

上に直接書いたのでこれも不要になる。

この記述の仕方には、最初は少し戸惑うかも知れません。そこで、少し補足しておきま

しょう。

- まず、イベントリスナとして用いるクラスは、BtnAction という名前で次のように宣言されていました。しかし、クラス名は何でもよく、必要なのは点線枠で囲まれた情報のみです。

```
class BtnAction implements ActionListener {
    public void actionPerformed(ActionEvent evt){
            jTxt.setText("成功！");
    }
}
```

- そこで、この点線枠部分のみを通常のクラスと同じように扱えるようにしようというのが Java 言語の考え方です。
- この点線枠の部分を**無名クラス**と呼びます。BtnAction というクラス名の情報を除外しているからです。
- そして、Java 言語では次の形でそのオブジェクトを生成することができます。

  new 無名クラス

- 一方イベントリスナの登録は

  jBtn.addActionListener(BAct);

  のように行われていました。そこで、この BAct の代わりに「new 無名クラス」部分を代入したものが上のプログラムです。
- 通常イベントリスナの登録はこのように行われます。

さて、このようにしてイベントリスナの登録部分は次のようになりました。最後に、これをもう一段拡張性の高い形にしておきましょう。

```
jBtn.addActionListener(new ActionListener {
    public void actionPerformed(ActionEvent evt){
        jTxt.setText("成功！");
    }
 });    //イベントリスナの登録
```

今の場合、イベント処理の内容は、「テキストフィールドに“成功！”という文字を表示する」という簡単なものなので気になりませんが、一般に処理内容が複雑になると、上の無名クラスの定義部分に書き込みと見通しが悪くなります。そこで、具体的な処理内容は所定のメソッドに記述する事にして次のように書き換えましょう。

```
jBtn.addActionListener(new ActionListener {
    public void actionPerformed(ActionEvent evt){
        jBtnActionPerformed(evt);
    }
 });    //イベントリスナの登録
```

新たにメソッドを導入

そして、新たに導入した jBtnActionPerformed()というメソッドを次のように定義します。

```
  void jBtnActionPerformed(ActionEvent evt){
     jTxt.setText("成功！");
  }
```

こうして、JavaAppli クラスの定義は最終的に次のようになりました。

```
public class JavaAppli {
   JButton jBtn;
   JFrame frame;
   JTextField jTxt;

  public static void main(String[] args) {
       . . .
   }

   void initGUI() {
       . . .
       jBtn.addActionListener(new ActionListener {
          public void actionPerformed(ActionEvent evt){
             jBtnActionPerformed(evt);
          }
       });    //イベントリスナの登録
   }

   void jBtnActionPerformed(ActionEvent evt){
        jTxt.setText("成功！");
   }

}
```

このように記述することにより、イベント処理の内容が変わった時には、所定のメソッド jBtnActionPerformed()の中身のみを変更すれば良く、他の部分は全く手を加える必要はなくなりました。

GUI Editor を用いてプログラムを作成する場合、ただ、jButton1ActionPerformed()などのメソッド内のみを記述すれば良いようになっていたのは、Eclipse が上のようにプログラムを記述してくれていたからです。そこで、改めて今まで作成したプログラムを眺めてみて下さい。一部理解できない部分があるかもしれませんが、概ね全体像を理解できるはずです。そうすれば、これまでより深く GUI Editor を含めた Eclipse を使いこなせるようになるはずです。そうすれば、初心者の域を超えてより専門的な次のステップへ進めることになります。