第５章　制御命令

【学習内容とねらい】

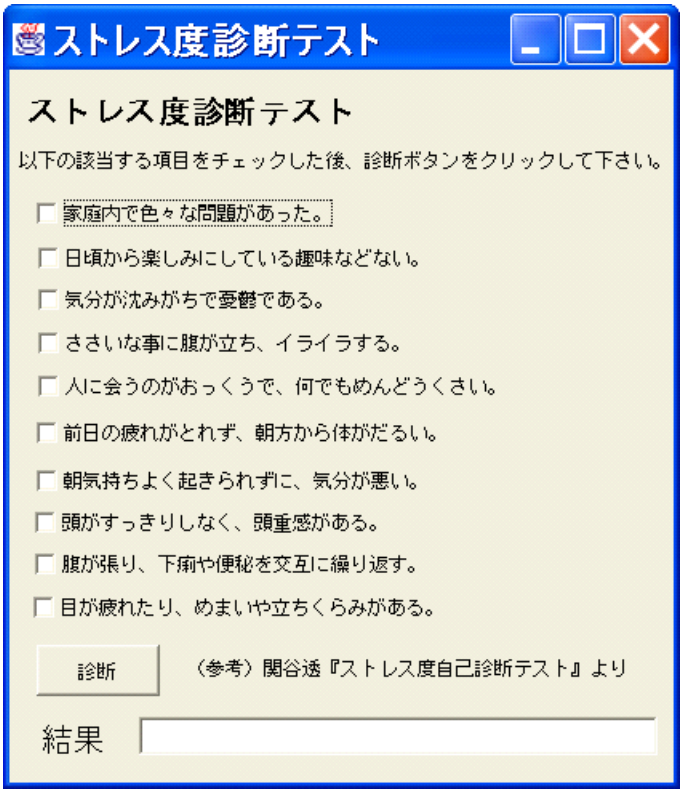
これまで学習したプログラムは、上から順番に１行ずつ処理を行うというものでした。これは処理の流れの基本形ですが、身の回りを見渡してみてもこれだけでは記述できない処理が多々あります。例えば、銀行のＡＴＭを考えてみましょう。この場合、ＡＴＭはインプットした暗証番号が正しいかどうかを判定して、正しければお金の引き出し等に応じ、間違っていれば処理を中断するようになっています。つまりある条件の判定結果に応じて処理内容を分けている（分岐させている）訳です。このような処理を「**分岐処理**」と言います。さらに、インプットした暗証番号が正しくなかった場合、ATM は番号が間違っている旨のメッセージを表示した上で、操作をやり直すために最初の画面に戻します。これは、操作を何度か繰り返す、つまり「**繰り返し処理**」に当たります。

プログラミング言語にも、当然この「分岐処理」と「繰り返し処理」の命令が用意されています。これらの命令は、処理を分岐したり、元に戻したりというように処理の流れを制御するので「**制御命令**」と呼ばれています。本章では、この「制御命令」の使い方を学習します。実は、どのような複雑な処理でも、この「分岐処理」と「繰り返し処理」の組み合わせによって記述できることが示されています。ですから、本章の学習を終えれば皆は、プログラミングに必要不可欠な“表現能力”を身に付けることになります。その意味で本章はプログラミング学習の“メインイベント”と言えるでしょう。どうか、張り切って学習に臨んで下さい。

＜第5章の構成＞

5-1 分岐処理（1）― 2分岐（if ～ else 文）―

以前、【基礎課題 2-5-3】で、下のようなフレームのプログラムを作りました。



このプログラムでは、

1. チェックされた項目の数を数える。
2. その数に応じて、診断結果を出す。

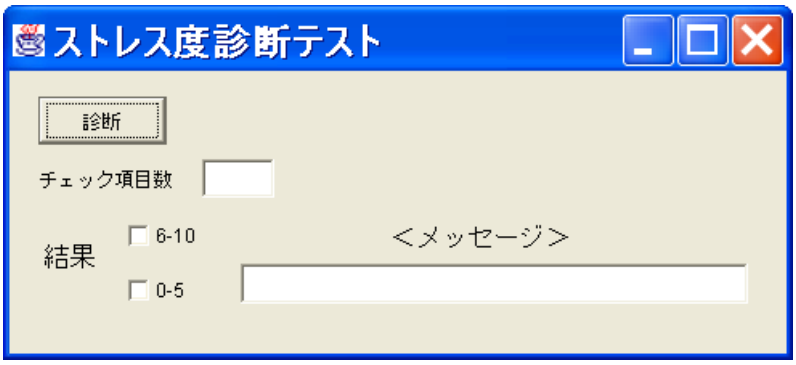
という２つの処理が必要です。

今回、このプログラムに、2. の機能をつけ加えてみましょう。

1. については、今回は考えません。項目の数は自分で数えることにします。

【基礎課題 5-1-1】

下のようなフレームを持つプログラムを作って下さい。



主なコンポーネントの name プロパティは、次のようにして下さい。

コンポーネント	name
ボタン「診断」	jButtonDiag
テキストフィールド「チェック項目数」	jTextFieldSum
チェックボックス「6-10」	jCheckBoxHigh
チェックボックス「0-5」	jCheckBoxLow
テキストフィールド「メッセージ」	jTextFieldMessage

「チェック項目数」を入力してから「診断」ボタンを押したとき、

- チェック項目数が 6 以上のとき
  - チェックボックス「6-10」をチェックする。
  - メッセージ欄に「少しストレスがたまっています。気分転換を。」と表示する。
- チェック項目数が 6 以上ではないとき
  - チェックボックス「0-5」をチェックする。
  - メッセージ欄に「ストレスの兆候がありますが、心配は不要。」と表示する。

という処理を行うプログラムを作りましょう。

ある条件が成り立っているか成り立っていないかによって行う内容が変わる場合には、「if 文」を用いて表します。

「診断」ボタンをクリックしたときのイベントハンドラを、次のように書いて下さい。

```
private void jButtonDiagActionPerformed(ActionEvent evt) {
  int sum=Integer.parseInt(jTextFieldSum.getText()); //項目数を変数宣言
  if (sum >= 6) {
    jCheckBoxHigh.setSelected(true); //上のチェックボックスをチェック
    jTextFieldMessage.setText("少しストレスがたまっています。気分転換を。");
  }
  else {
    jCheckBoxLow.setSelected(true);   //下のチェックボックスをチェック
    jTextFieldMessage.setText("ストレスの兆候がありますが、心配は不要。");
  }
}
```

字下げ　※字下げの幅はエディターの設定によって変わります。

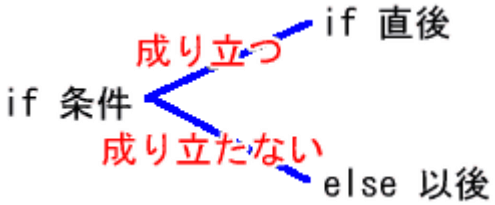
注意　字下げは Eclipse のエディターが自動的に行ってくれますので、それを利用して**必ず字下げを行いましょう**。構造が見やすくなります。

この命令は、下のような構造になっています。

```
private void jButtonDiagActionPerformed(ActionEvent evt) {
  int sum=Integer.parseInt(jTextFieldSum.getText()); //項目数を変数宣言
  もし                    が成り立つならば
  if (sum >= 6) {                                  成り立つ場合の処理
    jCheckBoxHigh.setSelected(true);
    jTextFieldMessage.setText("少しストレスがたまっています。気分転換を。");
  }
  そうでなければ
  else {                                           成り立たない場合の処理
    jCheckBoxLow.setSelected(true);
    jTextFieldMessage.setText("ストレスの兆候がありますが、心配は不要。");
  }
}
```

if 文の終わり

if 文は、条件が成り立っているか成り立っていないかによって、プログラムの流れを分岐させます。



条件は、

- 成り立つ
- 成り立たない

のどちらかしかありません。したがって、プログラムは、

- if 直後の文
- else 以下の文

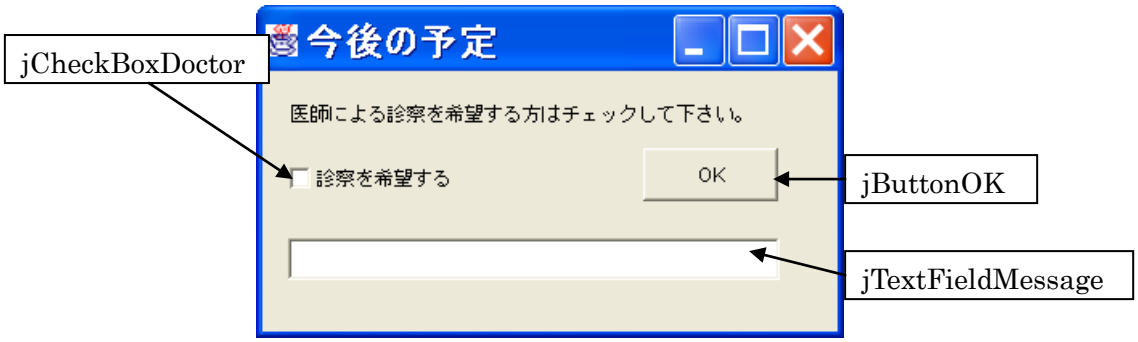
の**どちらかひとつ**を必ず実行し、かつ、いずれか**ひとつしか**実行しません。

Java 言語での不等号記号の書き方は右表の通りです。

数学表記	意味	Java での表記
A＞B	AはBより大きい	`A > B`
A≧B	AはB以上	`A >= B`
A＝B	AはBに等しい	`A == B`
A≦B	AはB以下	`A <= B`
A＜B	AはBより小さい	`A < B`
A≠B	AはBに等しくない	`A != B`

注意 Java 言語では、代入の「=」と比較の「==」を使い分けます。混同しやすいので注意してください。

【基礎課題 5-1-2】



上のようなフレームを作り、「診察を希望する」チェックボックスがチェックされているかどうかによって、テキストフィールド欄に

状態	メッセージ
チェックされている	来週水曜正午から、公民館で健康診断が行われます。
チェックされていない	ストレスを溜めない生活を心掛けましょう。

というメッセージが表示されるプログラムを作りましょう。
「OK」ボタンのイベントハンドラを次のように書いて下さい。

```
private void jButtonOKActionPerformed(ActionEvent evt) {
  if (jCheckBoxDoctor.isSelected()==true) {
    jTextFieldMessage.setText(
                        "来週水曜正午から、公民館で健康診断が行われます。");
  }
  else {
    jTextFieldMessage.setText("ストレスを溜めない生活を心がけましょう。");
  }
}
```

＜ 解説 ＞

① 「チェックボックスがチェックされているかどうか」は「selected」プロパティに入っています。そこでその値を得るに、は 4-11 節で学習した通り **isSelected()** というメソッドを用います。

さて、上の if 文では、

```
if (jCheckBoxDoctor.isSelected())
```

という様に、「==true」を省略する事ができます。 試しに == true を省略して実行してみましょう。うまく動きましたか？

コラム 「==true」の省略について

一般に if 文は次のように書けます。

```
if （論理式）
```

ここに論理式とは、（a>b）のように、それが成り立つか否か、つまり真（true）あるいは偽（false）のいずれかの値を持つ式のことです。

ところで、チェックボックス・コンポーネントの **selected** プロパティはそれ自身が true あるいは false のいずれかの値を持ちますから、これも論理式の一種と考えられます。そこで、

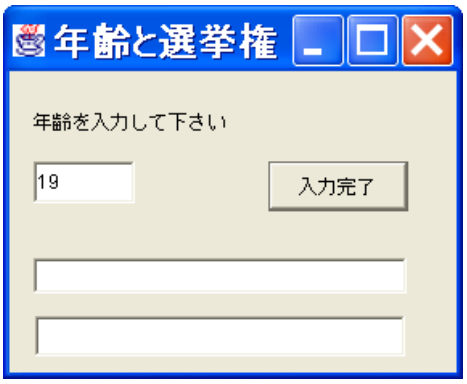
```
if (jCheckBoxDoctor.isSelected())
```

と書くことができるのです。つまり、論理型の変数の場合は==true をつける必要はありません。

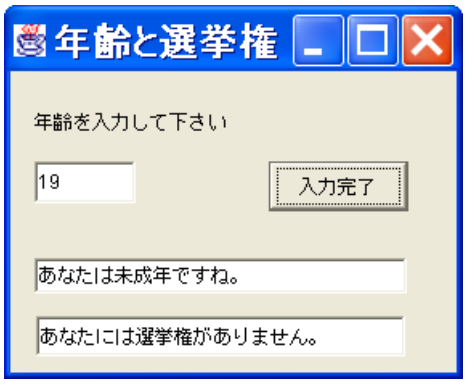
【基礎課題 5-1-3】

次のようなプログラムを作りましょう。

年齢を入力してから「入力完了」ボタンを押すと



２つのメッセージが表示される。



主なコンポーネントの name プロパティは、次のようにして下さい。

コンポーネント名	name
年齢用テキストフィールド	jTextFieldAge
ボタン「入力完了」	jButtonEnter
メッセージ用テキストフィールド（上）	jTextFieldM1
メッセージ用テキストフィールド（下）	jTextFieldM2

年齢とメッセージの対応は、次の表にしたがってください。

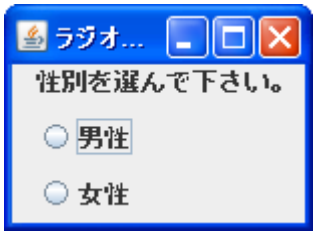
年齢	メッセージ
２０歳以上	あなたは成年ですね。あなたには選挙権があります。
１９歳以下	あなたは未成年ですね。あなたには選挙権がありません。

次の下線部を埋めて、プログラムを完成させて下さい。

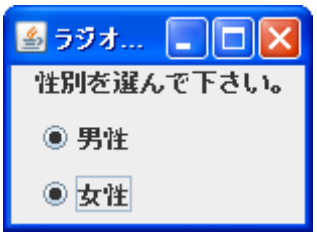
```
private void jButtonEnterActionPerformed(ActionEvent evt) {
   int Age=Integer.parseInt(jTextFieldAge.getText());
   if (Age >= _____ ) {
     _____;
     _____;
   }
   else {
     _____;
     _____;
   }
 }
```

ボタングループ（ButtonGroup）

次の課題に進む前に、新しいコンポーネント「ラジオボタン」の使い方を学習しておきましょう。ラジオボタンは、右のように Components タブの左から２番目にあります。

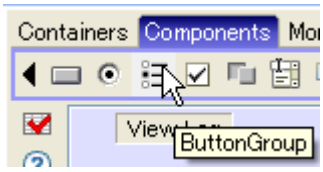


これを用いて次のようなプログラムを作ったとします。ラジオボタンはチェックボックスと同様、欄をクリックすることでチェックされた印が表示されるコンポーネントです。このフレームを作成することは難なくできると思います。



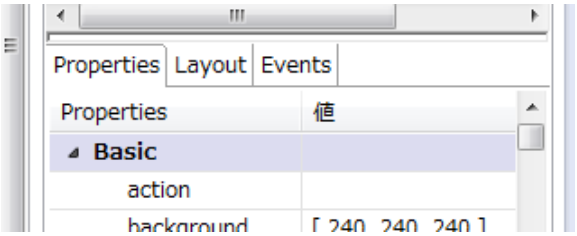
しかし、ただ、ラジオボタンを貼り付けただけでは、次のように実行時にどちらも（同時に）選択できるようになってしまいます。今の場合は、男性か女性か、どちらか一方しか選択できないようにしたいものです。

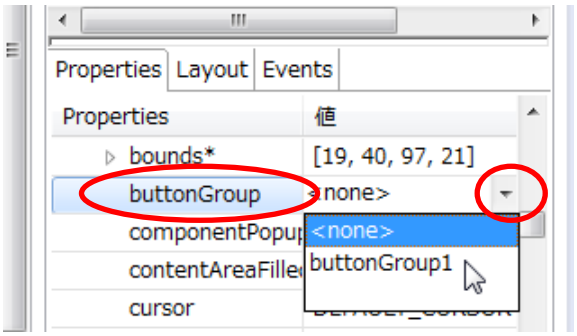
通常、Windows アプリケーションでは、ラジオボタンは幾つかの選択肢中から排他的にいずれか一つを選択させる場合に用います。実はそのようにするためには、**ボタングループ**という新しいコンポーネントが必要になります。



「ボタングループ」コンポーネントは、Components タグの左から３番目にあります。これをフレームに貼り付けてください。場所はどこでも結構です（ボタングループコンポーネントは表示されません）。

続いて、ラジオボタン（jRadioButton1）のプロパティを設定します。jRadioButton1 を選択した状態で、ワークベンチ画面右下の Properties 欄を見てください。





そしてプロパティ欄を下方にスクロールさせ、「**buttonGroup**」プロパティ欄を選択します（buttonGroup 欄は、**Basic** 項目の下の **Expert** という項目の中にあります）。ここに、「buttonGroup1」という、今貼り付けたボタングループコンポーネントが表示されるので、これを選択します。これで、ラジオボタン（jRadioButton1）はボタングループに登録されました。続いて、もう一つのラジオグループボタンも同様にボタングループに登録します。すると今度は、二つのラジオボタンの内、いずれか一方しか選択できないようになります。実はチェックボックスについても、同様です。以上を整理すると次のようになります。

＜まとめ＞

複数の選択肢からその中の一つを排他的に選択させたい場合には、次のようにします。

① 選択させるコンポーネント（ラジオボタンやチェックボックスなど）を貼り付けます。

② ボタングループコンポーネントを貼り付けます。

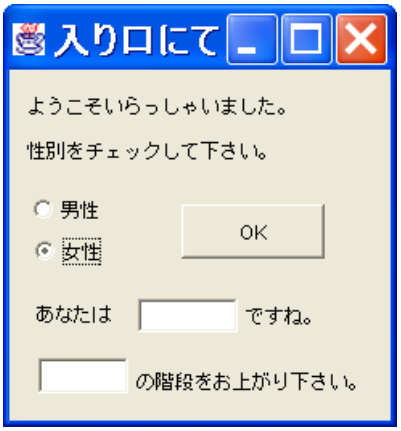
③ グループに入れるコンポーネントをボタングループに登録します。

なお、通常の Windows アプリケーションでは、チェックボックスは複数選択可に、ラジオボタンは排他的選択肢として使用します。

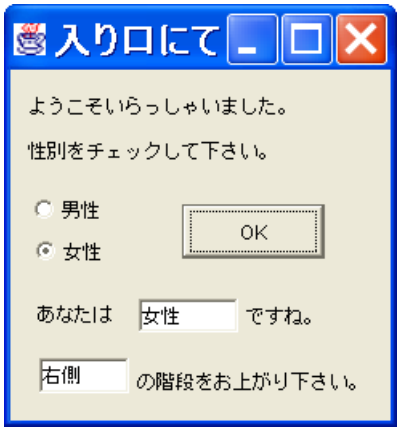
【基礎課題 5-1-4】

次のようなプログラムを作って下さい。

プログラムを起動し、性別を選択する（下は女性の例）。



［OK］ボタンをクリックすると、下のようなメッセージが表示される。



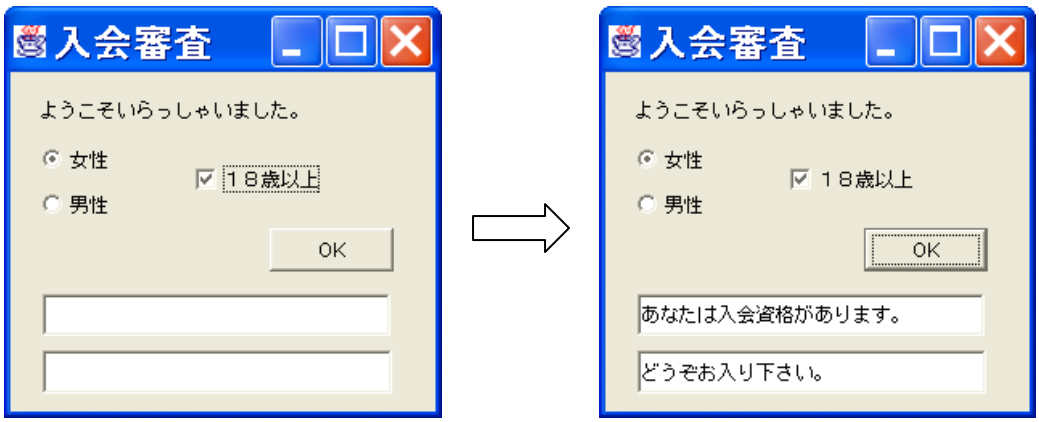
チェック項目とメッセージの対応は、次の表にしたがって下さい。

チェック項目	メッセージ
女性	あなたは女性ですね。右側の階段をお上り下さい。
男性	あなたは男性ですね。左側の階段をお上り下さい。

【基礎課題 5-1-5】

次のようなプログラムを作りましょう。

性別と年齢をチェックしてから「ＯＫ」ボタンを押すと、２つのメッセージが表示される。



主なコンポーネントの name プロパティは、次のようにして下さい。

コンポーネント名	name
ラジオボタン「女性」	jRadioButtonF
ラジオボタン「男性」	jRadioButtonM
ボタン「ＯＫ」	jButtonOK
チェックボックス「18 歳以上」	jCheckBoxAdult

２つのラジオボタンは、ボタングループに登録しています。

性別・年齢の条件とメッセージの対応は、次の表の通りです。

性別･年齢	メッセージ
女性で、かつ、１８歳以上	あなたは入会資格があります。どうぞお入り下さい。
それ以外	あなたは入会資格がありません。残念ですがお帰り下さい。

注意　「女性で、かつ、１８歳以上」つまり、
「jRadioButtonF がチェックされていて、**かつ**、jCheckBoxAdult がチェックされている」ことを、

```
(jRadioButtonF.isSelected() == true)  &&
(jCheckBoxAdult.isSelected() == true)
```

のように　**&&**　を用いて表します。
もちろん、先に学習した通り、== true を省略して

```
(jRadioButtonF.isSelected())  &&  (jCheckBoxAdult.isSelected())
```

と書くこともできます。通常はこちらの省略形の方が用いられるので、以下はこの記述を用いることにします。
　なお、&&　の前後の条件式（論理式）には、( ) をつけなくてもエラーにはなりません。しかし、条件式の区切りを明示的に示すために（　）でくくることを勧めます。なお、条件式全体のかっこ「if(条件式)」を外すと文法エラーとなりますので注意してください。

それでは、下の下線部を埋めて、プログラムを完成させて下さい。

```
private void jButtonOKActionPerformed(ActionEvent evt) {
  if ( (jRadioButtonF.isSelected()) &&
       (jCheckBoxAdult.isSelected()) ) {
    _____________________________________________________________;
    _____________________________________________________________;
  }
  else {
    _____________________________________________________________;
    _____________________________________________________________;
  }
}
```

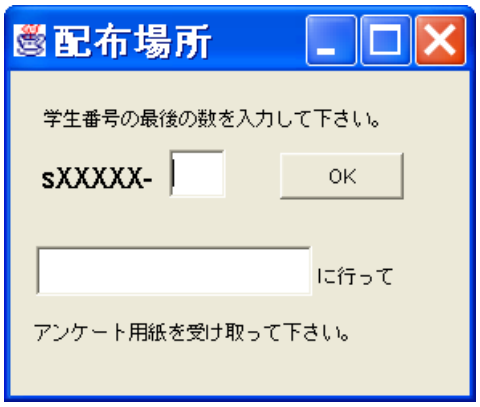
注意　「かつ」ではなく「**または**」を表したいときは、　**||**　を使います。

例　「女性、または、１８歳以上」のとき

```
(jRadioButtonF.isSelected())  ||
(jCheckBoxAdult.isSelected())
```

【基礎課題 5-1-6】

次のようなフレームを持つプログラムを作って下さい。



主なコンポーネントの name プロパティは、次のようにして下さい。

コンポーネント名	name
番号入力用テキストフィールド	jTextFieldNumber

番号入力欄には、０～９までの数しか入力しないものとします。
ここで、プログラムを実行し、数を入力してからＯＫボタンを押したとき、

- 数が 4 以上 7 以下であれば、メッセージ欄に「２０１号室」と表示する。
- 数がそれ以外であれば、メッセージ欄に「５０１号室」と表示する。

という処理を行うプログラムを作って下さい。

注意 今、番号入力欄に入力した値を、整数型変数 Num に次のように入力したとします。

```
int Num=Integer.parseInt(jTextFieldNumber.getText());
```

すると、入力された値が４以上７以下であることを表すのに、

```
if (4 <= Num <= 7)
```

と書きたくなると思います。しかし、Java 言語では、２つの不等号が同時に出てくる文を理解することができません。ですから、

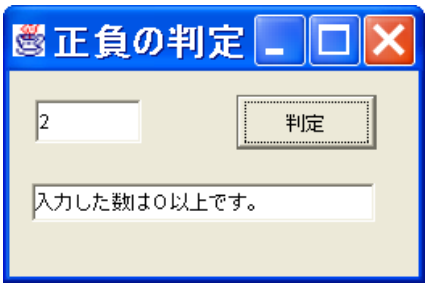
```
if ( (Num>=4) && (Num<=7) )
```

のように、条件を二つの部分に分けてから　&&（かつ）　でつなげて書く必要があります。この点に注意して下さい。

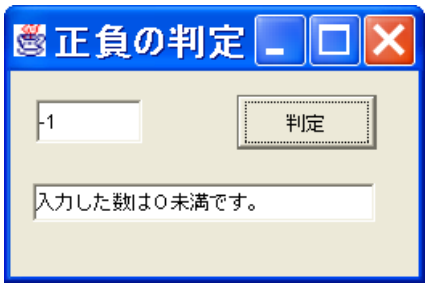
【練習問題】 if 文の{ }省略について

次のようなプログラムを考えましょう。

0 以上の数を入力してボタンを押すと、「入力した数は０以上です。」と表示する。



０未満の数を入力してボタンを押すと、「入力した数は０未満です。」と表示する。



コンポーネントの name プロパティは、次の通りとします。

コンポーネント名	name
入力用テキストフィールド	jTextFieldNumber
ボタン「判定」	jButtonHantei
判定欄用テキストフィールド	jTextFieldMessage

このとき、「判定」ボタンを押したときのイベントハンドラは以下のように書くことができます。

```
private void jButtonHanteiActionPerformed(ActionEvent evt) {
  int Num=Integer.parseInt(jTextFieldNumber.getText());
  if (Num>=0) {
    jTextFieldMessage.setText("入力した数は０以上です。");
  }
  else {
    jTextFieldMessage.setText("入力した数は０未満です。");
  }
}
```

ここで、if(条件式) のあと、あるいは else　のあとに、１つの処理（文）しかしないときは、{　}　を省略して、次のように書くこともできます。

```
private void jButtonHanteiActionPerformed(ActionEvent evt) {
  int Num=Integer.parseInt(jTextFieldNumber.getText());
  if (Num>=0) jTextFieldMessage.setText("入力した数は０以上です。");
  else        jTextFieldMessage.setText("入力した数は０未満です。");
}
```

しかし、実際 のプログラミングの場面では、最初は（処理する）実行文が１つでも、後に文を加えて複数の文になることがよくあります。その様な場合には再び｛　｝をつけ加えなければなりませんが、うっかり忘れてしまいやすいものです。そして、もし忘れるとエラーが発生したり思った通りの処理をしてくれなくなったりします。　そこで、処理する文が１つのときでも（{　}　を書いていてもエラーになるわけではないので)、{　}を記述することを勧めます。

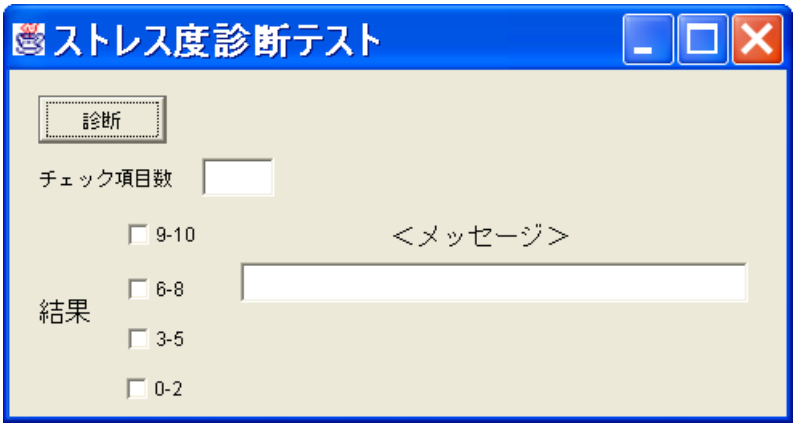
5-2 分岐処理 (2) —多分岐(if～else if～else 文)—

以前つくったストレス度診断プログラムは、もともと、下の分類にしたがって４段階の診断をするものでした。本節では、３つ以上の選択肢がある場合の処理を学習しましょう。

チェック数	メッセージ
９～１０	かなりストレスがたまっています。医師による診察を。
６～８	少しストレスがたまっています。気分転換を。
３～５	ストレスの兆候がありますが、心配は不要。
０～２	正常です。

【基礎課題 5-2-1】

【基礎課題 5-1-1】のプログラムを、次のように（選択肢を）拡張しましょう。



主なコンポーネントの name プロパティは、次のようにして下さい。

コンポーネント	name	備考
ボタン「診断」	jButtonDiag	変化なし
テキストフィールド「チェック項目数」	JTextFieldSum	変化なし
チェックボックス「9-10」	jCheckBoxHigher	
チェックボックス「6-8」	jCheckBoxHigh	
チェックボックス「3-5」	jCheckBoxLow	
チェックボックス「0-2」	jCheckBoxLower	
テキストフィールド「メッセージ」	jTextFieldMessage	変化なし

このように、３つ以上の分岐（選択肢）がある場合には、「if～else if～else 文」を用います。例えば４つの分岐がある場合には、次のように記述します。

```
if （条件式１） {
    条件式１が成立する場合の処理
}
else if （条件式２） {
    条件式２が成立する場合の処理
}
else if （条件式３） {
    条件式３が成立する場合の処理
}
else {
    上以外の場合の処理
}
```

「if～else if～else 文」の書き方

それでは、下の空欄を埋めて、４段階の診断をするプログラムを完成させて下さい。

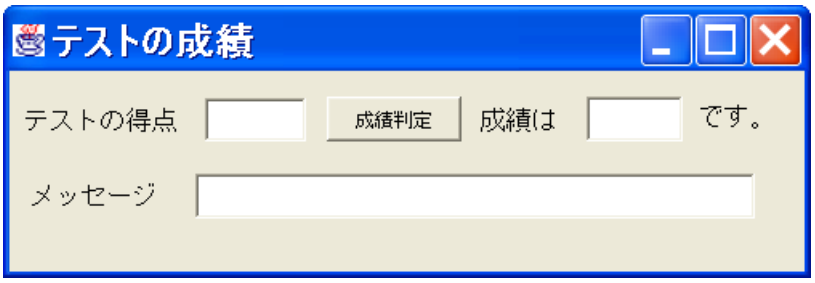
```
private void jButtonDiagActionPerformed(ActionEvent evt) {
  int sum=Integer.parseInt(jTextFieldSum.getText()); //項目数を変数宣言
  if (sum >= 9) {
      項目数≧9　の場合の処理
  }
  else if (sum>=6) {
      6≦項目数＜9　の場合の処理
  }
  else if (sum>=3) {
      3≦項目数＜6　の場合の処理
  }
  else {
      それ以外（今の場合　項目数＜3）　の場合の処理
  }
}
```

上の条件「項目数≧9」は（else により）除外されるので、自動的に「項目数＜9」という条件が付く事に注意。以下同じ。

※　上の例から理解できると思いますが、この「if～else if～else 文」はいくつ分岐があっても対応できます。

【基礎課題 5-2-2】

テストの得点を入力してから「成績判定」ボタンを押すと、得点に対応して「優」「良」「可」「不可」の評価をする下のようなプログラムを作って下さい。



得点と成績の対応は、次の表にしたがって下さい。

優	80 点～100 点
良	60 点～79 点
可	50 点～59 点
不可	0 点～49 点

また、成績に応じて、それぞれ内容の異なる一行メッセージを出すようにして下さい。メッセージの内容は自由に考えて下さい。

なお、「テストの得点」欄には 100 点満点のテストの得点を入力するものとします。

【基礎課題 5-2-3】

【基礎課題 5-2-1】のプログラムにおけるチェック数は、実は「3 で割ったときの商」によってメッセージが決まっています。

チェック数	3 で割った商	メッセージ
9～10	3	かなりストレスがたまっています。医師による診察を。
6～8	2	少しストレスがたまっています。気分転換を。
3～5	1	ストレスの兆候がありますが、心配は不要。
0～2	0	正常です。

したがって、予めチェック数を 3 で割って商を求めておくことによって、if 文の条件を書き直す事ができます。プログラムを次のように変更しましょう。空欄を埋めてプログラムを完成させてください。

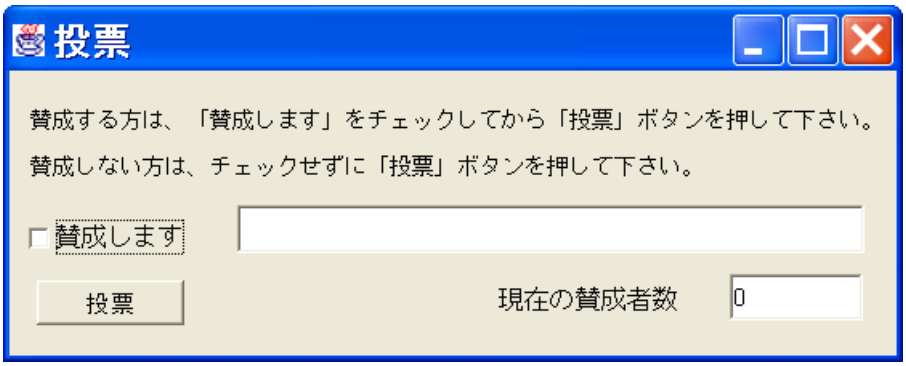
```
private void jButtonDiagActionPerformed(ActionEvent evt) {
  int sum=Integer.parseInt(jTextFieldSum.getText()); //項目数を変数宣言
  int q=sum/3;  //項目数を３で割った値（商）としてｑを定義
  if (q==3) {
    q=3 の場合（項目数≧9 の場合）の処理
  }
  else if (q==2) {
    q=2 の場合（6≦項目数＜9 の場合）の処理
  }
  else if (q==1) {
    q=1 の場合（3≦項目数＜6 の場合）の処理
  }
  else if (q==0) {
    q=0 の場合（項目数＜3 の場合）の処理
  }
}
```

※ 太枠の部分は、今の場合「else 」だけに替えても結構です。ただし、厳密に言うと、その際は「ｑの値が０～３以外の値をとらない」という前提が必要になります（今の場合はこの前提が成り立ちます）。

5-3 分岐処理（3）―2分岐の特殊な形（if～文）―

【練習問題】

下のようなフレームのプログラムを考えます。



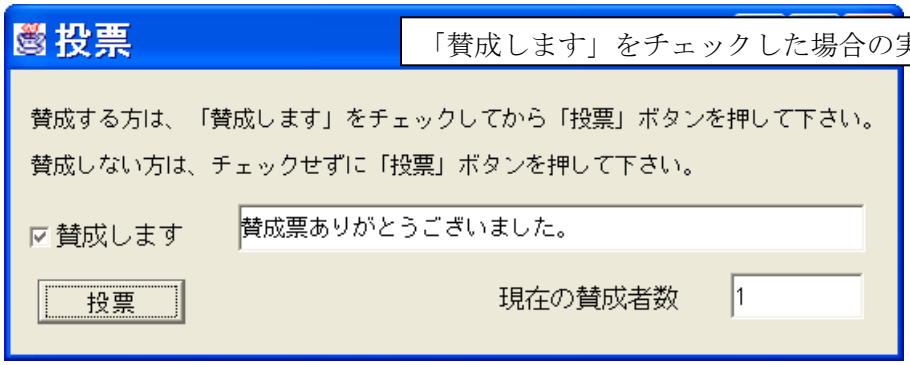
主なコンポーネントの name プロパティは、次の通りとします。

コンポーネント	name
チェックボックス「賛成します」	jCheckBoxSansei
ボタン「投票」	jButtonTouhyou
テキストフィールド「メッセージ」	jTextFieldMessage
テキストフィールド「現在の賛成者数」	jTextFieldNinzu

さて、プログラムを実行して、「投票」ボタンを押したとき、

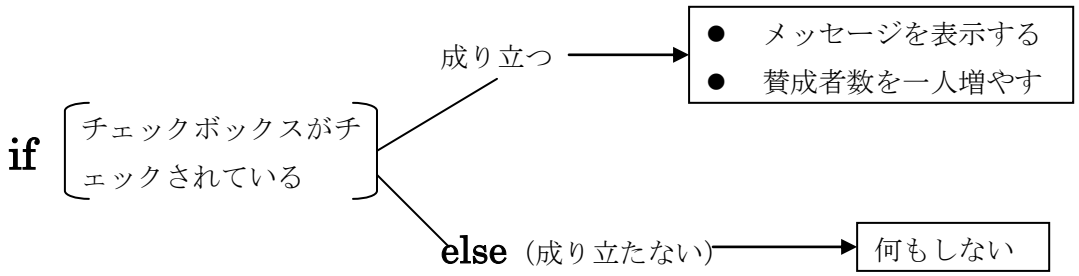
- もしチェックボックスがチェックされていれば、
  - 「賛成票ありがとうございました」というメッセージを表示し、
  - 賛成者数を１人増やす。
- もしチェックボックスがチェックされていなければ、
  - 何もしない。

というようなプログラムを作ります。



「賛成します」をチェックした場合の実行結果

このプログラムの流れを図にすると、次のようになります。



プログラムは、次のようになります。

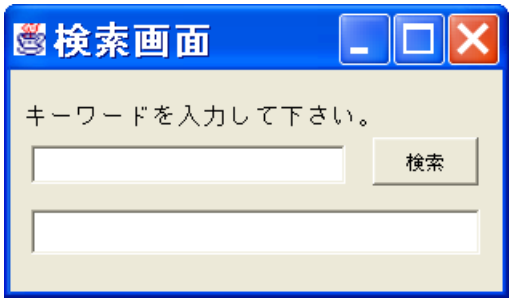
```
private void jButtonTouhyouActionPerformed(ActionEvent evt) {
  //賛成者人数を保管する変数の定義
  int Ninzu=Integer.parseInt(jTextFieldNinzu.getText());
  if (jCheckBoxSansei.isSelected()) {
    jTextFieldMessage.setText("賛成票ありがとうございました。");
    //人数を一人増やして表示
    jTextFieldNinzu.setText(String.valueOf(Ninzu+1));
  }
  else {
            何もしないので何も書かない
  }                                                    省略できる
```

解説

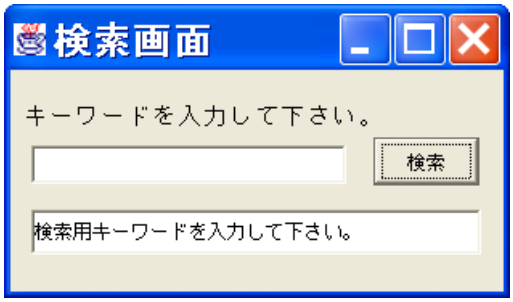
- 条件が成り立たない場合には何もしないので、**else** のあとの｛　　｝の間には、何も書きません。
- このような場合は、**else** と、そのあとの｛　　｝を省略するのが普通です。

【基礎課題 5-3-1】

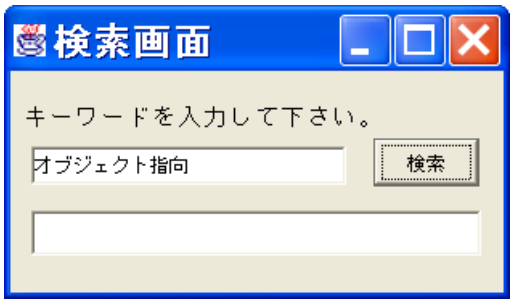
次のようなプログラムを作って下さい。作成するプログラムは検索処理を模したプログラムです。ただし、現時点では残念ながら検索処理をプログラミングすることはできません。



プログラムを起動すると、左のような画面が現れます。



キーワードを入力しないでボタンを押すと、「検索用キーワードを入力してください。」と警告し、



キーワードを入力してボタンを押すと何もしない。

コンポーネントの name プロパティは次の通りとします。

コンポーネント	name プロパティ
テキストフィールド「キーワード」	jTextFieldKey
テキストフィールド「メッセージ」	jTextFieldMessage
ボタン［検索］	jButtonFind

次の下線部を埋めてプログラムを完成させて下さい（次ページ解説参照）。

```
void jButtonFind_actionPerformed(ActionEvent e) {
  if (jTextFieldKey.getText().equals("")) {
    ____________________________________________________________;
  }
}
```

＜ 解説 ＞

① 何も入力していない状態とは、テキストフィールドの text プロパティが**空文字**""に等しい状態です。

② すると、「jTextFieldKey が空の場合」という条件は、

```
if (jTextFieldKey.getText()=="")
```

と表されそうですが、実はこれではうまく行きません。整数型や論理型変数と違って、文字列型同士の比較の場合は、上のプログラムのように **equals()** メソッドを用いなければならないのです。→　下の補足参照

③ else 文以下は、何もしないので省略しています。

補足　メソッドを持つということから推測されるように、文字列型（String）は、実は単なる”（変数の）型”ではなく**クラス**です。したがって、String 型変数は、String クラスの**オブジェクト**ということになります。上の例では jTextFieldKey.getText()がそのオブジェクトに当たります。そして、オブジェクト同士の比較の「==」を用いると、それはそれぞれのオブジェクトがメモリ上に保管されている場所（アドレス）の比較という、特別の意味を持つのです。つまり、文字列の値そのものの比較ではないので、上の「jTextFieldKey.getText()==""」という条件式は成立することがないのです。

文字列１　==　文字列２ ⇒ 成立することはない

「文字列１の中身の保管場所」≠「文字列２の中身の保管場所」

少しややこしいですが、現時点では、文字列型の比較だけ特別である、とだけ頭に入れて下さい。

【応用課題 5-3-A】

上のプログラムでは、一度警告メッセージが出てしまうと、それ以降は（例えキーワードを入力して［検索］ボタンを押しても）警告が消えてくれません。本来は、キーワードを入力してボタンを押すと、その警告は消えるべきですね。そこで、そのように【基礎課題 5-3-1】を改良して下さい。

ヒント　次の二通りの考え方があります。

① ［検索］ボタンを押したとき、まず（無条件で）メッセージ欄を消す（空文字を代入する）ようにする。

② ［検索］ボタンを押したとき、キーワードが入力されている場合にはメッセージ欄を消す（空文字を代入する）ようにする。

5-4 分岐処理（4）－多分岐（switch 文）－

最も基本的な if 文は、 5-1 節で学習した「if～else 文」であり、それは

- 条件が成り立つ（**then**）
- 条件が成り立たない（**else**）

に応じて、プログラムの流れを２つに分岐させるものでした。

プログラムの流れを３つ以上に分けなければならないときは、5-2 節で学習したように、「if～else if～else～文」を用います。これで、あらゆる分岐処理を記述することができるのですが、多分岐の場合、Java 言語には **switch** 文が用意されており、これを使うとプログラムの記述が容易になり、かつ見易くなる場合があります。そこで、本節では、この **switch** 文の使い方を学習しましょう。

【基礎課題 5-2-3】は、**switch** 文を使うと次のように書くことができます。

```
private void jButtonDiagActionPerformed(ActionEvent evt) {
  int sum=Integer.parseInt(jTextFieldSum.getText()); //項目数を変数宣言
  int q=sum/3;  //項目数を３で割った値（商）
  switch (q) {
    case 3:
        q ＝3　の時の処理
      break;
    case 2:
        q ＝2　の時の処理
      break;
    case 1:
        q ＝1　の時の処理
      break;
    case 0:
        q ＝0　の時の処理
      break;
  }
}
```

注意　switch 文で分岐に使用する変数（上の例では q ）は、整数型（int 型）でなければなりません（文字型 **char** 型も使用できますが本テキストでは扱いません）。

【基礎課題 5-4-1】

前ページの空欄（各 q の値に対応する処理）を埋めて、【基礎課題 5-2-3】のプログラムを書き換え、動作を確認してください。

switch 文を用いる際、最も注意しなければならない点は、**break** 文の位置です。例えば前ページのプログラムにおいて、一番上の break 文を削除すると・・・

```
switch (q) {
    case 3:
        q=3 の時の処理
    case 2:
        q=2 の時の処理
        break;
    case 1:
        q=1 の時の処理
        break;
    case 0:
        q=0 の時の処理
        break;
}
```

q ＝ 3 の場合、「q=3 の処理」を行った後、「q=2 の処理」も実行します。

＜break 文の役割＞

switch 文の中では、break 文を見つけると、switch 文ブロックの外に出ます。一方、break 文がないと次の case へ次々進み、break 文を見つけるまで、それ以降の処理を実行します。

この性質を利用して、変数の範囲に応じて分岐を行うことができます。
例えば、整数 q の値が「０～１」および「２～３」に応じて処理を分岐させたい場合には、次の様に記述します。

```
switch (q) {
    case 3:
    case 2:
        q=2～3 の時の処理
        break;
    case 1:
    case 0:
        q=0～1 の時の処理
        break;
}
```

＜変数の範囲を指定する場合＞

【基礎課題 5-4-2】

【基礎課題 5-2-1】を、switch 文を使って作り直してください。ただし、商(q)を用いずに、項目数(sum)を直接用います。これは、前ページで説明した break 文を活用する練習です。

＜default を用いた記述＞

switch 文には、**default** を使った特殊な例があります。一番下の「変数の値の範囲」を default と書いて、「今までに指定した範囲以外」という意味を表すことができます。

例えば、今ある変数 q の値が１～１０までの値をとるものとします。そして、以下の３つの場合に処理を分岐させたい場合を考えましょう。

① 変数 q ＝ 1 の時の処理
② 変数 q ＝ 5 の時の処理
③ それ以外の時の処理

このような場合、逐一変数の範囲を指定すると、③の処理の指定が煩雑になりますが、**default** 文を用いると次のように簡単に記述できます。

```
switch (q) {
    case 1:
        ①　q＝1　の時の処理
        break;
    case 5:
        ②　q＝5　の時の処理
        break;
    default:
        ③　上記以外の時（今の場合 q ＝2～4,6～10　の時）の処理
        break;
}
```

※　もし必要がない場合（例えば③の場合は何も処理をしない場合）には、default 文以下を省略することができます。

【応用課題 5-4-A】

【基礎課題 5-4-2】のプログラムでは、項目数として１１以上の値や、負の値など、想定している範囲外の値を入力した場合、何もメッセージが表示されません。そこで、このような範囲外の項目数を誤って入力してボタンをクリックした場合は、「０～１０までの項目数を入力してください。」というメッセージを表示させるように、このプログラムを改良してください。default を用いれば簡単にプログラミングできるはずです。

5-5 繰り返し処理（1）―累乗―

スーパーファミコンは 16 ビット、PlayStation は 32 ビット、NINTENDO 64 は 64 ビット、さらにはコンピュータのメモリは 64 MB あるいは 128 MB 等々、コンピュータ関係のもので出てくる数には決まった数が多いですね。これらの数は、2×2×2×…×2 の形で求められる、コンピュータにとって“きりのいい”数なのです。

【練習問題】

下の表を埋めましょう。

		2 の個数	答（計算結果）
2^1	2	1	2
2^2	2×2	2	4
2^3	2×2×2	3	
2^4	2×2×2×2	4	
2^5	2×2×2×2×2	5	
2^6	2×2×2×2×2×2	6	
2^7	2×2×2×2×2×2×2	7	
2^8	2×2×2×2×2×2×2×2	8	
2^9	2×2×2×2×2×2×2×2×2	9	
2^{10}	2×2×2×2×2×2×2×2×2×2	10	

2×2×2×2×2 のように、ある数　a を b 個かけることを累乗といいます。
2×2×2×2×2 のことを、2 を 5 個書く代わりに「2^5」と書いて「2 の 5 乗（じょう）」と読みます。

【基礎課題　5-5-1】

プログラムを起動し、「2 倍」ボタンを押したら

欄内の 2 倍になり、さらにもう 1 度「2 倍」ボタンを押したら

さらに 2 倍になり、ボタンを押すごとに値がどんどん 2 倍になっていく、というプログラムを作りましょう。

配置するコンポーネントの name プロパティは次のようにして下さい。

コンポーネント	name
テキストフィールド	jTextFieldNumber
ボタン	jButtonDouble

注意　テキストフィールドの欄内の初期値を 1 にしておいてください。

次の下線部を埋めてプログラムを完成させて下さい。

```
private void jButtonDoubleActionPerformed(ActionEvent evt) {
  //欄内の値を整数型変数 Num として定義
  int Num=Integer.parseInt(jTextFieldNumber.getText());
  Num=_____________________________;
  jTextFieldNumber.setText(String.valueOf(Num));
}
```

ヒント　現在の欄内の値（Num）を２倍すれば良いのです。これと同じ処理を加算（＋）について、【基礎課題 4-10-1】で行いました。

【練習問題】

このプログラムを使えば、2^5 の値を調べたいときも「2 倍」ボタンを 5 回押すだけで求められます。では、このプログラムをもっと改良して、ボタンを 1 度押すだけで 2^5 が求められるようにできないでしょうか。

これはそれほど難しくありません。ボタンを押したときに今までの 5 回分の動作をすればよいのですから、

```
private void jButtonDoubleActionPerformed(ActionEvent evt) {
  //欄内の値を整数型変数 Num として定義
  int Num=Integer.parseInt(jTextFieldNumber.getText());
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  jTextFieldNumber.setText(String.valueOf(Num));
}
```

5 回分の処理

となります。実行して動作を確認してみましょう。

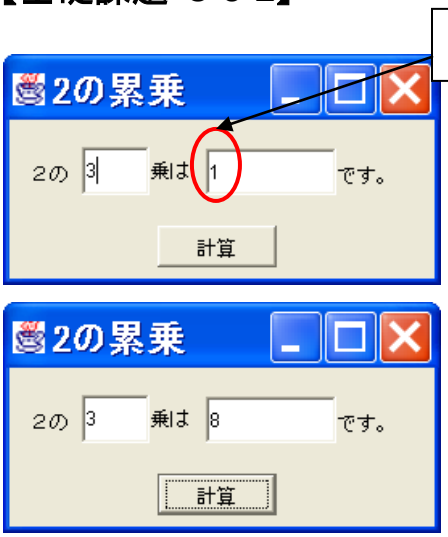
【基礎課題 5-5-2】

ボタンを 1 度押すだけで 2^{15} が求められるよう、プログラムを改変して下さい。

```
private void jButtonDoubleActionPerformed(ActionEvent evt) {
  //欄内の値を整数型変数 Num として定義
  int Num=Integer.parseInt(jTextFieldNumber.getText());









  jTextFieldNumber.setText(String.valueOf(Num));
}
```

5-6 繰り返し処理（2）ーfor 文の導入ー

【基礎課題 5-6-1】



何乗の値を計算したいか（ここでは 3 乗）を入力して「計算」ボタンを押すと

その値を計算してくれる、というプログラムを作りましょう。

コンポーネントの name プロパティは次のようにします。

コンポーネント	name
左のテキストフィールド	jTextFieldJo
右のテキストフィールド	jTextFieldNumber
ボタン	jButtonCalc

このプログラムでは、「何回 2 倍するか」はプログラム実行後にユーザが決めるので、今までのようにプログラム実行前に

```
private void jButtonCalcActionPerformed(ActionEvent evt) {
  int Num=Integer.parseInt(jTextFieldNumber.getText());
  Num=Num*2;
  Num=Num*2;
  ...
      続く
  jTextFieldNumber.setText(String.valueOf(Num));
}
```

とあらかじめ決まった回数だけ書いておくことはできません。
そこで、プログラム中で繰り返し回数を指定できる「**for** 文」という命令を使います。
プログラムを次のように書いてください。

```
private void jButtonCalcActionPerformed(ActionEvent evt) {
  int Jo=Integer.parseInt(jTextFieldJo.getText());
  int Num=Integer.parseInt(jTextFieldNumber.getText());
  int i;
  for (i=1; i<=Jo; i=i+1) {
    Num=Num*2;
  }
  jTextFieldNumber.setText(String.valueOf(Num));
}
```

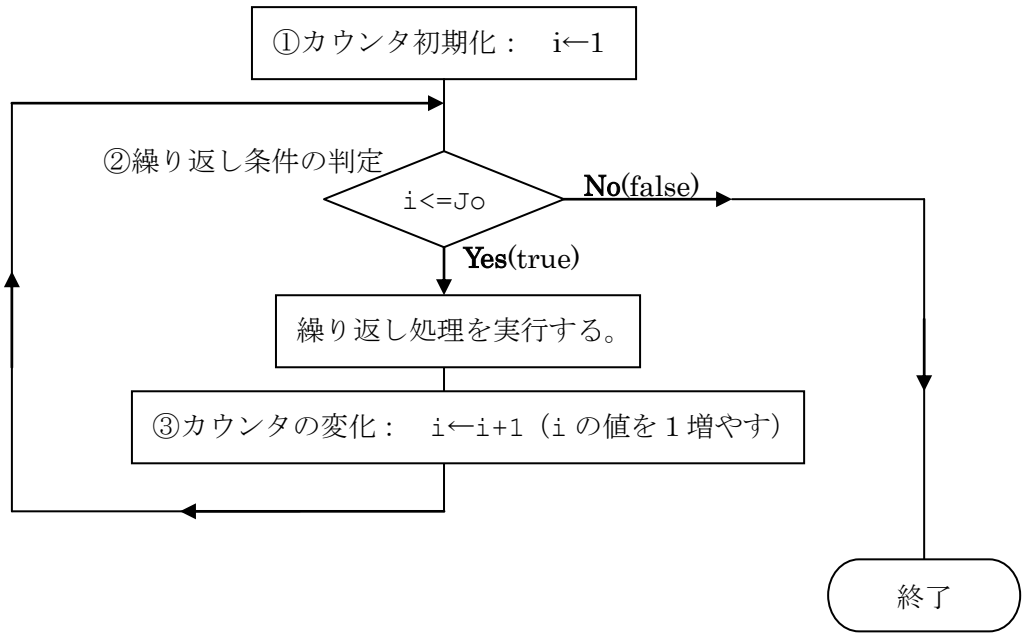
何回目かを数える変数（**カウンタ**）の宣言

繰り返したい処理

実行して動作を確認しましょう。この **for** 文は、下のような構造になっています。

```
                                  ①          ②          ③
for ( カウンタ（上の例では i）の初期化; 繰り返し条件;カウンタの変化式 ) {
    繰り返し処理１;
    繰り返し処理２;
       ・・・
}
```

意味はだいたい推測できると思いますが、処理の流れを以下に示しておきましょう。



なお、Java 言語では、上のプログラムは通常、次の様に（簡略化した形で）記述されます。

```
 private void jButtonCalcActionPerformed(ActionEvent evt) {
   int Jo=Integer.parseInt(jTextFieldJo.getText());
   int Num=Integer.parseInt(jTextFieldNumber.getText());
   for (int i=1; i<=Jo; i++) {
     Num=Num*2;
   }
   jTextFieldNumber.setText(String.valueOf(Num));
 }
```

波線部については、4-6 節コラム（変数の初期化）で説明した「変数の宣言と初期化は同時に行える」こと、また、4-7 節（様々な演算子）で説明した「++」演算子を用いています。このように、for 文は決まった回数の繰り返しに用いられます。下の左右のプログラムは全く同じ動作をします。

```
void jButtonCalc_actionPerformed
(ActionEvent e) {
  int Num=1;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
  Num=Num*2;
}
```

１０回

```
void jButtonCalc_actionPerformed
(ActionEvent e) {
  int Num=1;
  for (int i=1; i<=10; i++) {
      Num=Num*2;
  }
}
```

コラム　{ }の省略について

```
for ( カウンタ用変数の初期化; 繰り返し条件;カウンタの変化式 ) {
    繰り返し処理１;
}
```

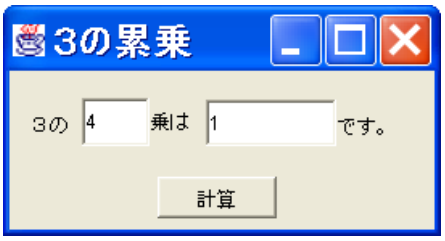
というように、繰り返し処理が１つしかない場合は、if 文の場合と同様に、{ } を省略して次のように書くことが出来ます。

```
for ( カウンタ用変数の初期化; 繰り返し条件;カウンタの変化式 )
    繰り返し処理１;
```

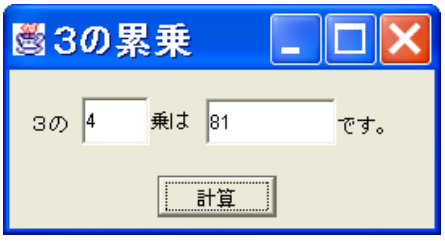
しかし、この書き方は、後で繰り返し処理を増やしたくなった場合に { } を付け忘れる可能性があり、実際これまでもそのようなエラーは（本演習において）頻繁に見受けられました。そこで、たとえ処理が１つでも常に { }をつけることを勧めます。

【基礎課題 5-6-2】

先ほどのプログラムを改変して、



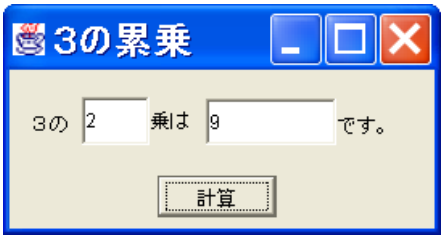
数値を入れたら（この例では ４）



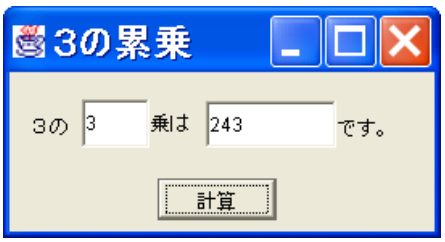
3 の累乗が求められるプログラムを作ってください。

【応用課題 5-6-A】

【基礎課題 5-6-2】で作ったプログラムでは、



一度、３の累乗を求め（この例では、3^2）



その後続けて、次の計算（この例では 3^3）を行おうとすると、正しく求まりません。左の例では、以前の結果である９から計算がスタートするので、答えは、$3^2 \times 3^3$になってしまいます。

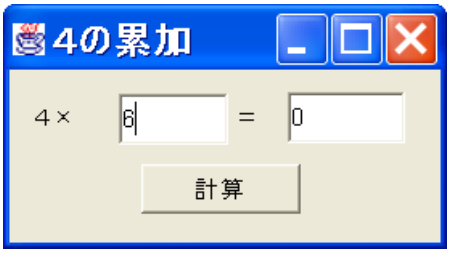
そこで、一々答えの欄を１にリセットしなくても続けて３の累乗が求められるようにプログラムを改良してください。

ヒント　【基礎課題 5-6-2】のプログラムでは、計算結果を Num という整数型変数に入れておきましたが、この Num の値を変数宣言時に１に初期化するようにすれば良いのです。

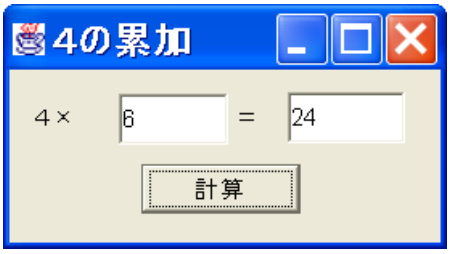
【基礎課題 5-6-3】

かけ算は、足し算の繰り返しと考えることができます。例えば、２×５は、「２を足す」ということを５回繰り返すことです。

このように考えると、かけ算の記号「*」を使わずに、for 文を使ってかけ算の答を求めることができます。そこで、次のようなプログラムを作って下さい。

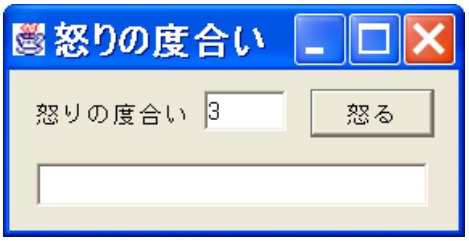


数値を入れたら（この場合は 6）

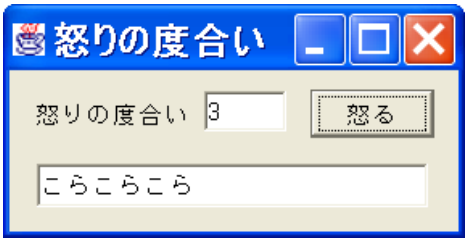


4+4+4+…+4 を求めるプログラムを、for 文を使って作って下さい（ただし、プログラムの中でかけ算「*」を使ってはいけません）。

【基礎課題 5-6-4】



怒りの度合いを数値で入力し「怒る」ボタンを押すと



その度合いに応じて「こらこら…」と表示するプログラムを作って下さい。

- 度合いが 1 なら「こら」
- 度合いが 2 なら「こらこら」
- 度合いが 3 なら「こらこらこら」

(以下同様)

コンポーネントの name プロパティは次の通りとします。

コンポーネント	name プロパティ
テキストフィールド「怒りの度合い」	jTextFieldDoai
テキストフィールド「表示欄」	jTextFieldResult
ボタン	jButtonAnger

次の下線部を埋めてプログラムを完成させて下さい。

```
private void jButton1ActionPerformed(ActionEvent evt) {
  int Doai=Integer.parseInt(jTextFieldDoai.getText());
  String Result=""; //表示欄の文字列の初期化（空文字にしておく）
  for (int i=1;  __________  ;i++) {
    Result=_____________________;
  }
  jTextFieldResult.setText(Result);
}
```

注意 jTextFieldResult の text プロパティに入る文字列を文字列型変数「**Result**」として宣言しています。

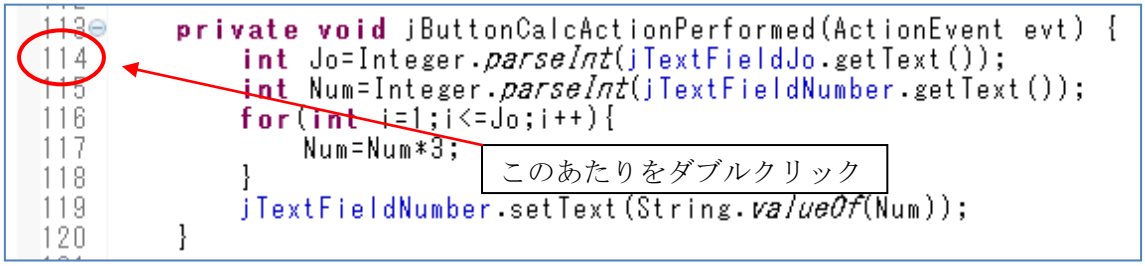
ヒント ‘こら’という文字列を、指定回数だけ連結すれば良いのです。

5-7 繰り返し処理（3） —for 文の流れの観察(デバッガ利用)—

【練習問題】

【基礎課題 5-6-2】で作った「３ の累乗」プログラムで「計算」ボタンを押したときのプログラムが動作する様子を１行ずつ順に追いかけてみましょう。そのために Eclipse に備わっているデバッグ機能（デバッガ）を用いることにします。

まずは、実行時に一旦プログラムを止めるため、「ブレークポイント（“一時停止”の標識）」をおきます。次のようにコードエディタの”int Jo=･･･”の行番号部分をダブルクリックしてください（作成手順の違いによって行番号はこれと微妙にずれている場合があります）。

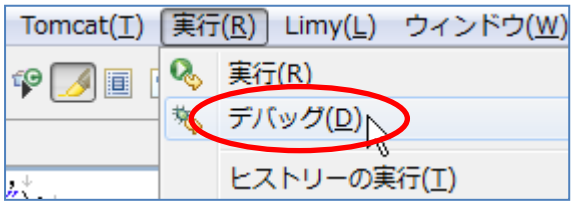



```
113    private void jButtonCalcActionPerformed(ActionEvent evt) {
114        int Jo=Integer.parseInt(jTextFieldJo.getText());
115        int Num=Integer.parseInt(jTextFieldNumber.getText());
116        for(int i=1;i<=Jo;i++){
117            Num=Num*3;
118        }
119        jTextFieldNumber.setText(String.valueOf(Num));
120    }
```

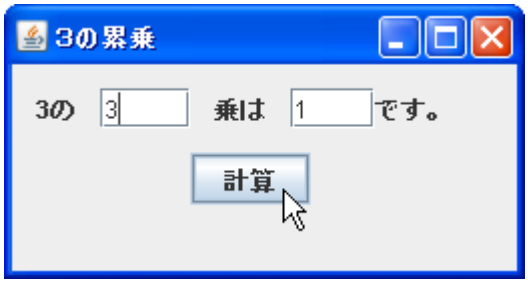
すると、ブレークポイント（小さな青丸）が現れます。この位置でプログラムが停止します。

```
113    private void jButtonCalcActionPerformed(ActionEvent evt) {
114        int Jo=Integer.parseInt(jTextFieldJo.getText());
115        int Num=Integer.parseInt(jTextFieldNumber.getText());
116        for(int i=1;i<=Jo;i++){
117            Num=Num*3;
118        }
119        jTextFieldNumber.setText(String.valueOf(Num));
120    }
```

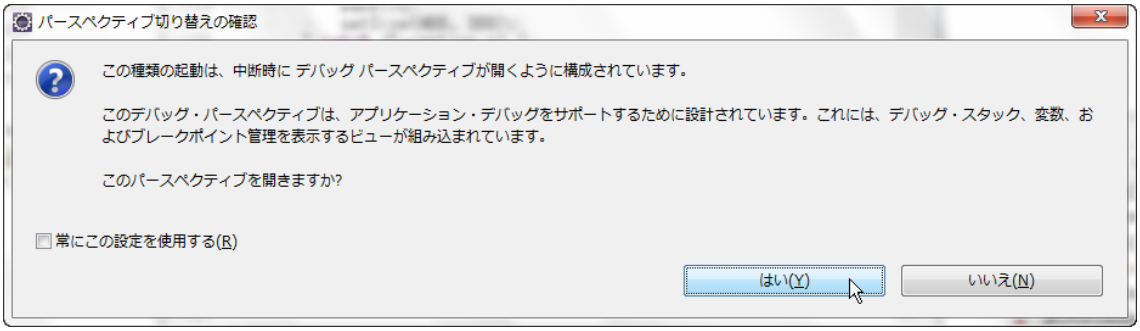
さて、それでは、デバッグに取りかかりましょう。ワークベンチの「実行」メニューから「デバッグ」を選択します。



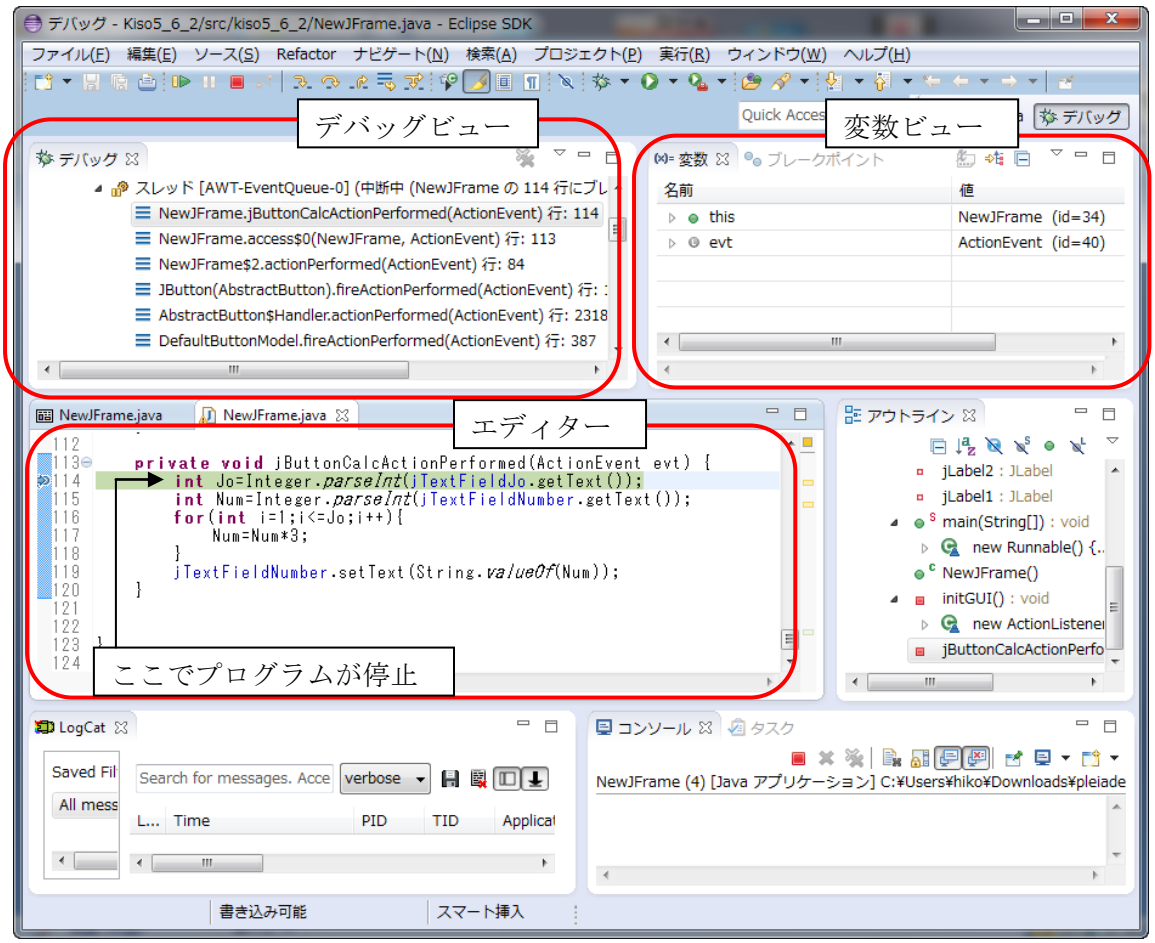
すると、いつものようにプログラムが起動します。そして、右のように入力して［計算］ボタンを押すと・・・、



次の様にデバッグパースペクティブへの切り換えを確認する画面が出てきます。ここでは、［はい］を選んでください。

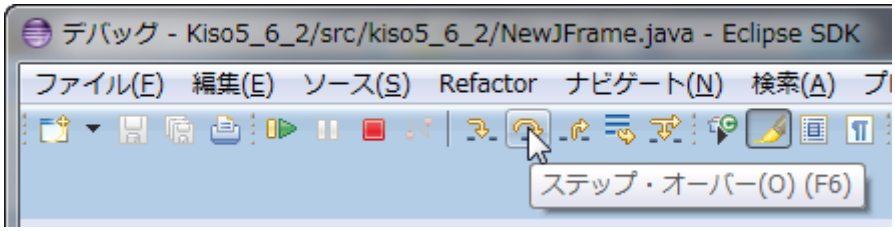


すると、ワークベンチが、デバッグ用の画面（パースペクティブ）に変わります。

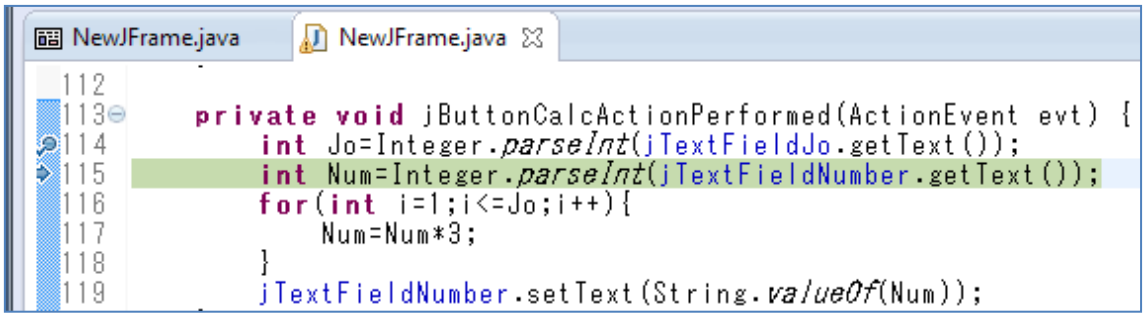


エディターを見ると、先ほど指定したブレークポイントでプログラムが停止しているはずです。

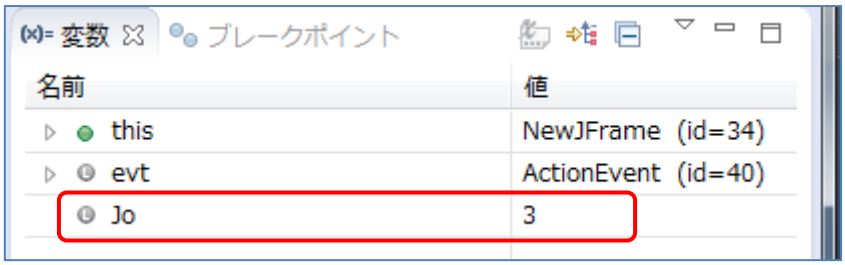
今から１行ずつプログラムを実行させて行くのですが、そのためには、次のように画面上方のメニュー欄にある、「ステップオーバー」を選択します。



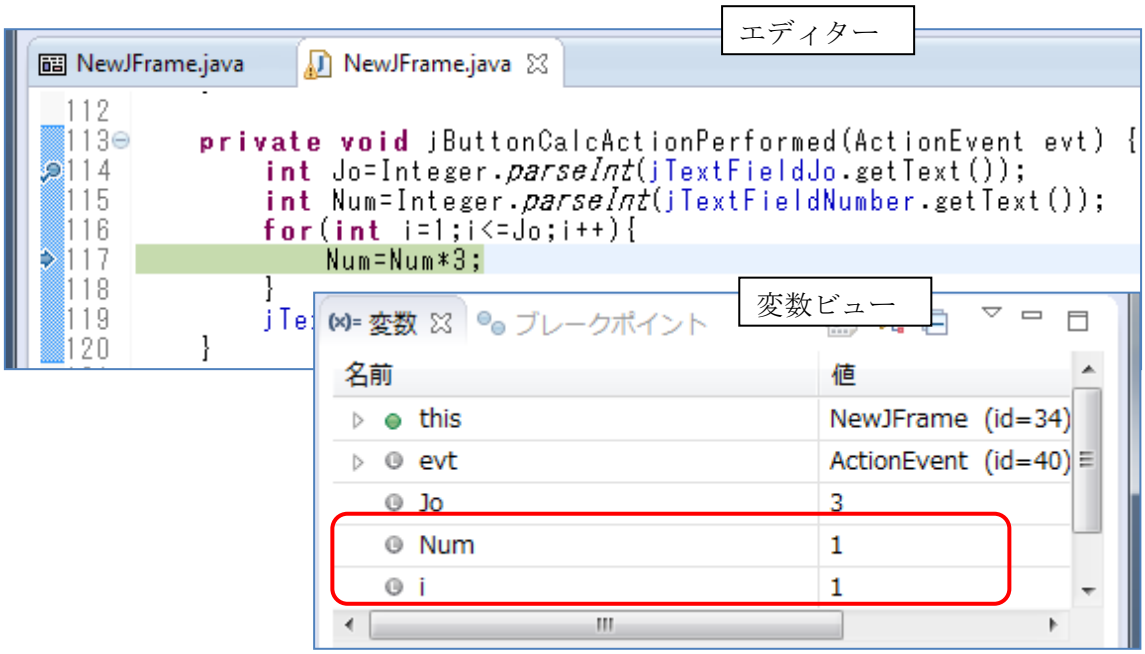
この「ステップオーバー」ボタンをクリックする毎に、次のように 1 行ずつプログラムが進行します（下は１回クリックして実行箇所が１行下に移動したところ)。



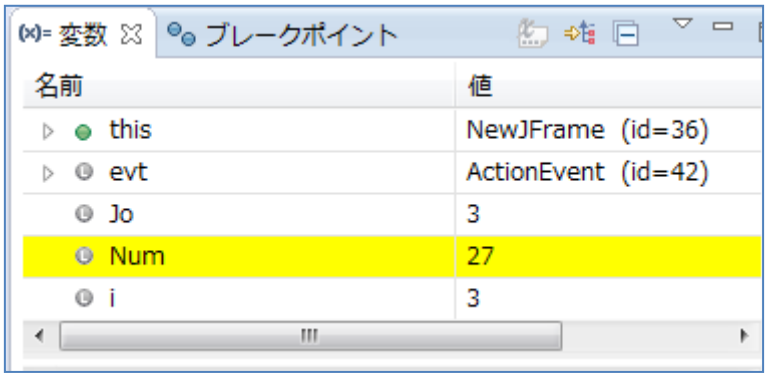
このとき、「変数」ビューを見ると、変数「Jo」が現れ、その値が「３」になっています。これは、テキストフィールドに入力した値である「３」が変数 Jo に代入されたためです。



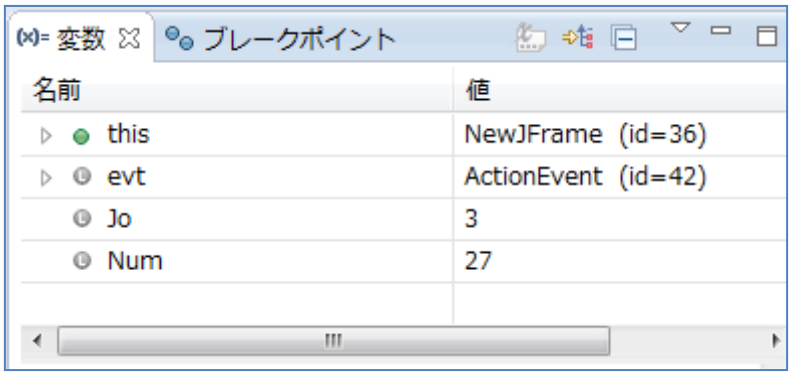
さらにもう２回「ステップオーバー」ボタンをクリックすると、次のようになります。



ステップオーバーを続けて行くと次のようになります。

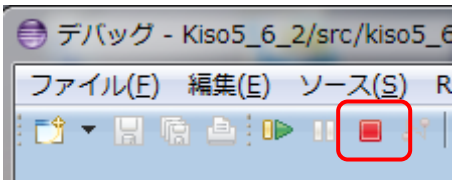


繰り返し処理を 3 回行ったところ。繰り返し回数を示す変数 i の値が 3 になっている。このとき、変数 Num の値は 3^3=27 になっている。

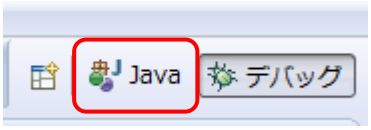


（for 文の）繰り返し処理のループを抜け出してプログラムが終了したところ。for 文のブロックを抜け出したので、（ブロック内で宣言された）変数 i は消えている。

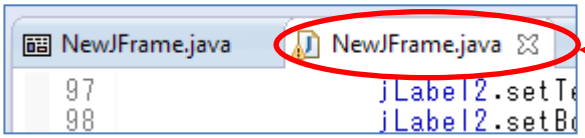
上のようにプログラムの最終行まで達したら、次のように、デバッグビューの終了ボタンをクリックします。これでデバッグは終了です。



ワークベンチを元の Java パースペクティブに戻すため、ワークベンチ右上隅のパースペクティブ選択欄から「Java」パースペクティブを選択します。これで、Java パースペクティブに切り替わります。



Java パースペクティブに戻ると、デバッグしたプログラムの編集画面がもう一つ開かれています。これは、デバッグ作業用に開かれた画面なのでデバッグ終了後は閉じます。



こちらを閉じる。

【基礎課題 5-7-1】

上のデバッガを用いて、3^5を求める過程で、１回の繰り返しループが終了する毎に変数 i と Num がどのように変化するかを確認し、以下の表に記入して下さい。

i	1				
Num	3				

コラム　デバッグとは？

デバッグとは debug と書き、本来の意味は「ムシとり」つまり害虫駆除です。bug（バグ）とはムシのことなのです。

プログラムを作成する際にエラーはつきものです。スペルミスなど単純な文法エラーの場合は、すぐに見つけ出せますが、思いこみや勘違いなどによるエラーはプログラマ本人でも（むしろ本人であるためなおさら）なかなか見つけ出せません。それは、作物や衣類等大切なものに忍び込んだムシ（害虫）のようにやっかいなものです。そのような事から、いつの頃からかプログラマ達はエラーのことを、バグ（ムシ）と呼ぶようになりました。そしてエラーを見つけ出してそれを修正する作業をデバッグ（ムシとり：害虫駆除）という様になったのです。プログラマの間ではよく、「またバグ（ムシ）が見つかったよ。」とか、「今度のバグは手強いぞ！」等という会話がなされます。何となくユーモラスですね。

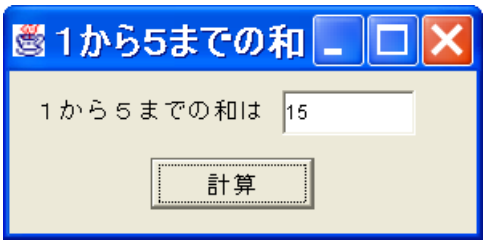
さて、エラーのないプログラムを書くようにするのは理想です。しかし、そこは人間のなせる技ですから、エラーはつきもの。そこで、エラーを見つけ出し、それを修正するデバッグの技術を身につけることが実際には大変重要になるのです。最も確実なデバッグ手法は、プログラムの処理を１行ずつ追いかけて行って、どこで不具合が生じているかを見つけ出す手法です。これを**トレース**（追跡）と言います。上で利用したデバッガの機能はトレースを支援する機能だったのです。トレースができるようになったら確実にプログラミングの力がつきます。皆も、プログラムの実行結果に何か問題があった時にはトレースするよう心がけて下さい。

5-8 繰り返し処理（4）-カウント用変数を使ったプログラム-

i は何回目の繰り返しかを数える変数です。これを文法用語では「制御変数」と呼びますが、ここでは分かりやすく「カウント用変数（**カウンタ**）」と呼びましょう。5-6 節のプログラム例では、{　} の間を 1 回実行するたびに、i は 1 つずつ増えます。この性質を利用すると、自然数の和などを簡単に求めることができます

【基礎課題 5-8-1】

次のように、[計算] ボタンを押すと、1 から 5 までの和（1+2+3+4+5）を求め表示するプログラムを作りましょう。

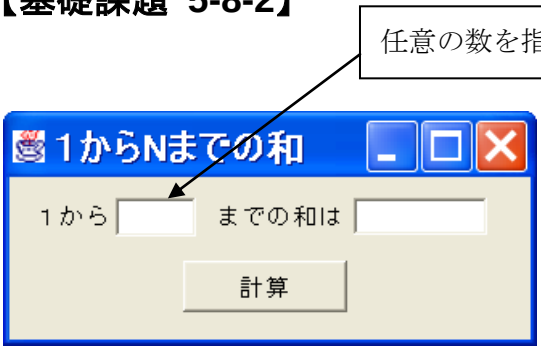


コンポーネント	name
テキストフィールド	jTextFieldSum
ボタン	jButtonCalc

下線部を埋めてプログラムを完成させて下さい。

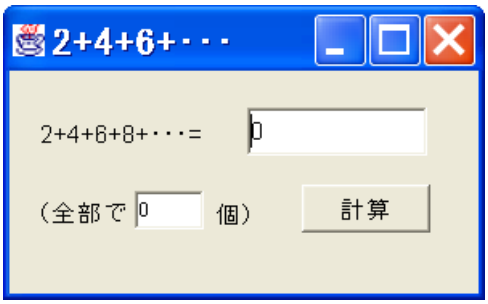
```
private void jButtonCalcActionPerformed(ActionEvent evt) {
  int Sum=0; //合計を保管する変数 Sum を初期化
  for (int i=1; i<=5; i++) {
    Sum = Sum + _________;
  }
  jTextFieldSum.setText(String.valueOf(Sum));
}
```

【基礎課題 5-8-2】

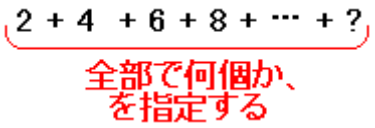


上のプログラムを改良して、5 までではなく任意の数までの和を求める（どこまで足すかを指定できる）プログラムを、for 文を使って作って下さい。

【基礎課題 5-8-3】



2+4+6+8+… を計算するプログラムを、for 文を使って作って下さい。

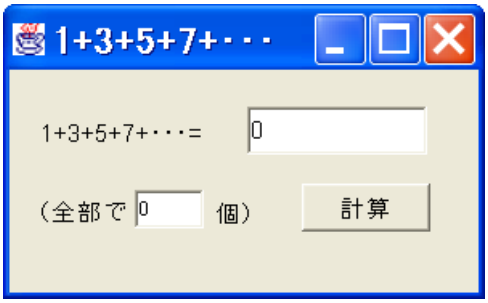


ヒント　例えば、４個の和を求める場合、カウンタ i と加えるべき数との対応は次の通りです。

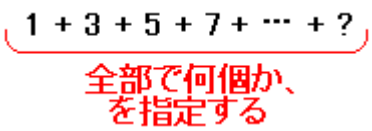
カウンタ i	1	2	3	4
加えるべき数	2	4	6	8

すると、「加えるべき数」は i 用いてどのように表されるでしょうか？そこがポイントです。

【基礎課題 5-8-4】

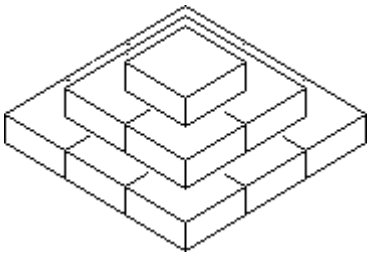


1+3+5+7+・・・ を計算するプログラムを、for 文を使って作って下さい。

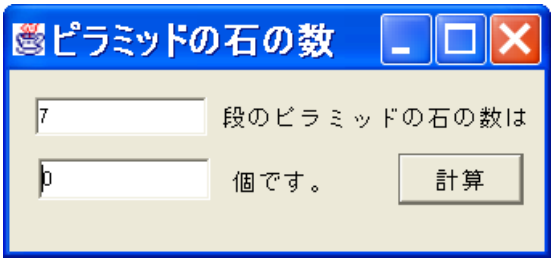


ヒント　前課題同様、加えるべき数をカウンタ i でどのように表されるか、がポイントです。

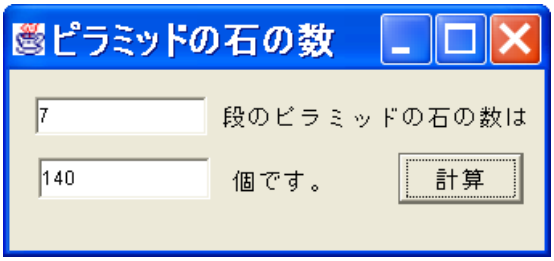
【応用課題 5-8-A】



左のピラミッドは、1 番上の段は石が $1^2=1$ 個、上から 2 段目は $2^2=4$ 個、上から 3 段目は $3^2=9$ 個です。このピラミッドが 5 段になったときの石の個数を求めるプログラムを作って下さい。
また、このプログラムを改良して、次のようなプログラムを作って下さい。



ピラミッドの段数を入力して「計算」ボタンをクリックしたら



答えを出力するプログラムを作って下さい。

【応用課題 5-8-B】

右は、かけ算の九九表です。この全ての数の合計を求めるプログラムを作って下さい。

1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

ヒント　例えば、１行目（１番上の横の並び）の和を求めることはすでにできるはずです。これに、２行目、３行目の和を順次加えて行けばよいのです。

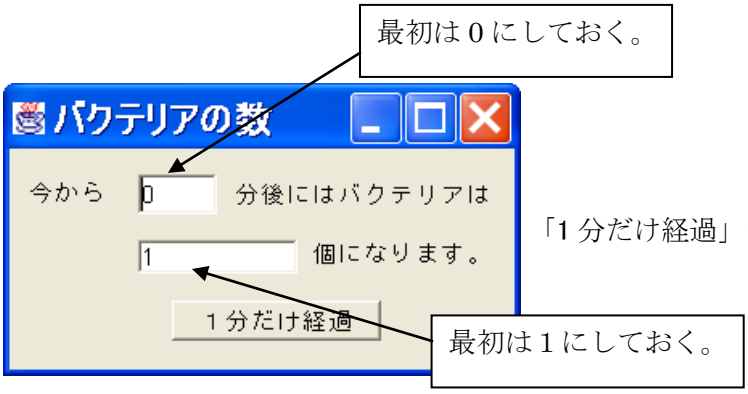
すると・・・

最終的には「行を数える for 文の中に列を数える for 文が入る」という形になるはずです。
これができたら大したもの！

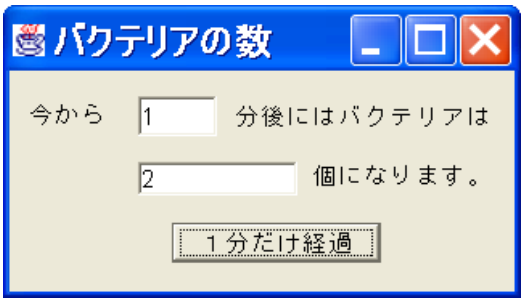
5-9 繰り返し処理（5）―回数が定まっていない繰り返し―

1 分間で 2 個に分裂するバクテリアが、今 1 個あります。このバクテリアが 10000 個以上になるのは何分後でしょうか。

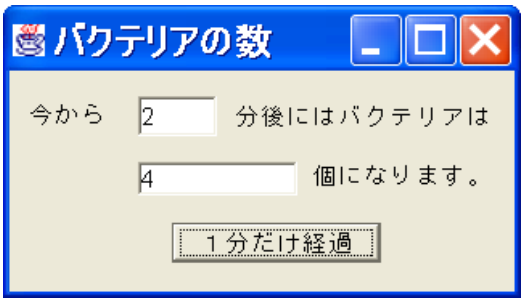
この問題を解くため、



「1 分だけ経過」ボタンを押すと



1 分経過してバクテリアが 2 倍になり、さらに「1 分だけ経過」ボタンを押していくと



バクテリアがどんどん増えていき、いつかは 10000 個を越える、というプログラムを作ります。

【基礎課題 5-9-1】

上のプログラムを作りましょう。コンポーネントの name プロパティは次の通りとします。

コンポーネント	name
上のテキストフィールド（分後）	jTextFieldMinute
下のテキストフィールド（個）	jTextFieldNum
ボタン	jButtonCalc

ボタンを押したときのイベントハンドラは次のようになります。下線部を埋めてプログラムを完成させて下さい。

```
private void jButtonCalcActionPerformed(ActionEvent evt) {
        //（経過）分を保管する変数の宣言と初期化
  int Minute=Integer.parseInt(jTextFieldMinute.getText());
        //バクテリアの個数を保管する変数の宣言と初期化
  int Num=Integer.parseInt(jTextFieldNum.getText());
  Minute= _____;
  Num= _____;
  jTextFieldMinute.setText(String.valueOf(Minute)); //分の表示
  jTextFieldNum.setText(String.valueOf(Num)); //個数の表示
}
```

また、このプログラムを用いて、バクテリアが 10000 個以上になるのは何分後かを求めて下さい。

_____分後

5-10 繰り返し処理（6） —While 文の導入—

前のプログラムでは、何度もボタンを押していくとそのうち答えが求まります。でも、ボタンを１回押すだけで答えが求まるプログラムにしたいですね。
何回もボタンを押すことは、for 文で出てきた「繰り返し」と呼ばれる操作（作業）です。ですから、まずは for 文を使ってボタンを１回押すだけのプログラムを考えてみましょう。プログラムは次のようになります。

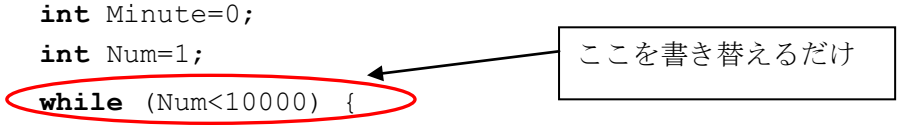
```
private void jButtonCalcActionPerformed(ActionEvent evt) {
  int Minute=0; //（経過）分の初期化（0 分から始まるので）
  int Num=1;  //個数の初期化（１個からスタートしているので）
  for (int i=1; Num<10000 ; i++) {
     Minute= Minute+1; //１分経過
     Num=Num*2 ;  //バクテリアの数を２倍する
  }
  jTextFieldMinute.setText(String.valueOf(Minute)); //分の表示
  jTextFieldNum.setText(String.valueOf(Num)); //個数の表示
}
```

今の場合、繰り返し条件を下線部の通りとすれば良いことは理解できると思います。これで OK なのですが、全く使わない変数（カウンタ）i を宣言し１ずつ増やすのは何か無駄ですね!?　今の場合カウンタは必要ありません。そこで、カウンタを用いない繰り返し文の書き方を次に学習しましょう。

【練習問題】

カウンタを必要としない繰り返しには、「while 文」を使います。プログラムを次のように変更してください。

```
private void jButtonCalcActionPerformed(ActionEvent evt) {
  int Minute=0;
  int Num=1;
  while (Num<10000) {
    Minute=Minute+1;
    Num=Num*2;
  }
  jTextFieldMinute.setText(String.valueOf(Minute));
  jTextFieldNum.setText(String.valueOf(Num));
}
```



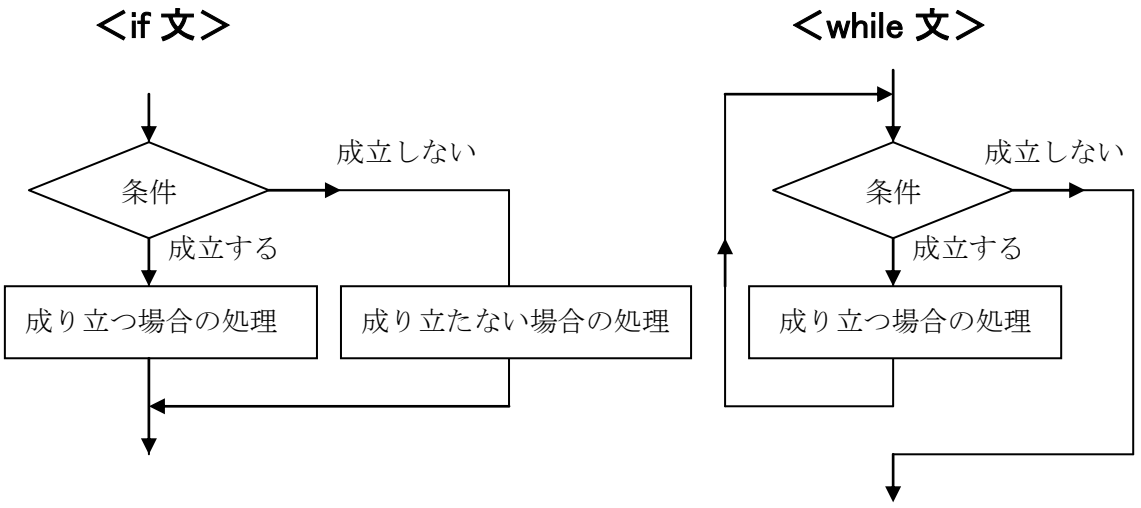
このプログラムの処理内容は上の for 文によるプログラムと全く同じです。実は、**for** 文あるいは **while** 文のいずれか一つだけでも、あらゆる繰り返し処理を実現できます。一般には、繰り返し回数などカウンタが必要になる場合は **for** 文、それ以外の場合は **while** 文を用いると良いでしょう。

while 文は、「条件が成立している間は処理を繰り返す」という制御を行い、右のように書きます。

```
while(条件) {
        繰り返し処理 1;
        繰り返し処理 2;
              .
              .
}
```

プログラムを実行して、結果を確かめてみましょう。ちゃんと「14 分」という答が出ましたか?

条件によって処理が変わるという点で if 文と while 文は似ていますが、if 文は条件が成立してもしなくても 1 回きり、while 文は条件が成立している間何度でも繰り返す点が異なります。



コラム　{　}の省略について

```
while(条件) {
    繰り返し処理;
}
```

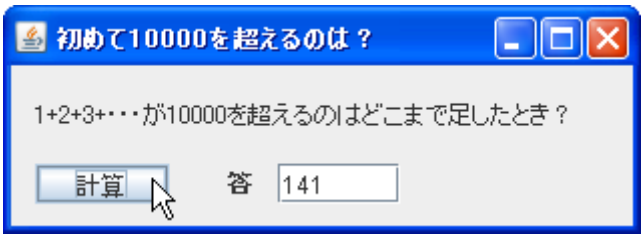
というように、繰り返し処理が 1 つしかない **while** 文は、**if** 文あるいは **for** 文の場合と同様に、{　} を省略して次のように書くことが出来ます。

```
while(条件)
  繰り返し処理;
```

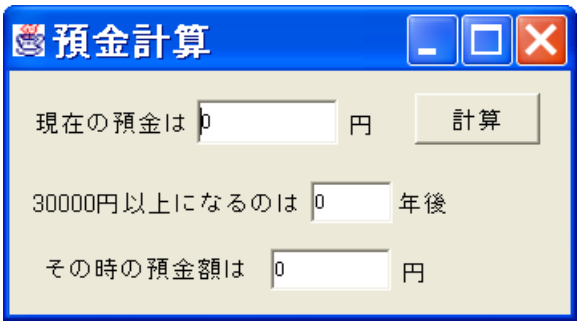
しかし、この書き方では、後で繰り返し処理を増やしたくなった場合に {　}を付け忘れる可能性があります。これまで同様、たとえ処理が 1 つでも常に {　}をつけることを勧めます。

【基礎課題 5-10-1】

1+2+3+4+･･･ と足していくと、はじめて 10000 を超えるのはどこまで足したときでしょうか? 次のようなプログラムを使って求めて下さい。(数列の和の公式 $\frac{n(n+1)}{2}$ は使わないで下さい。)



【応用課題 5-10-A】



1 年間に 5% の利子がつく（つまり預金が 1.05 倍になる）口座があります（今となっては夢のような利率ですが・・・）。この口座に 10000 円を預けておくと、何年後に 30000 円以上になるでしょうか? 右のようなプログラムを作って答を求めて下さい。

動作内容は次の通りです。

① 現在の預金を入力し［計算］ボタンを押す。

② すると、30000 円以上になるのが何年後か、およびその時の預金総額を表示する。

なお、毎年の利子に端数（円以下）が出ますが、全て小数点以下は切り捨てとして下さい。それには、4-7 節で学習した「実数型→整数型」の**型キャスト**を用います。