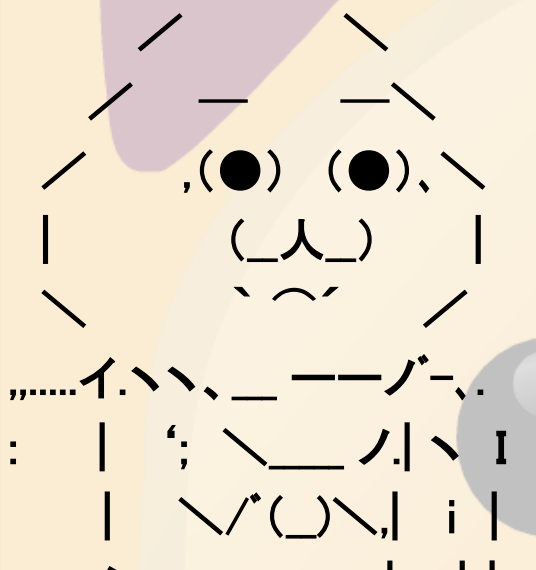
匠の伝承w

マルチな時代の設計と開発

パート5

スピーカー自己紹介



ゆーちです。

ハンドル名です。

本名は、内山康広といいます。

48歳です。

おっさんです。_|￣|○

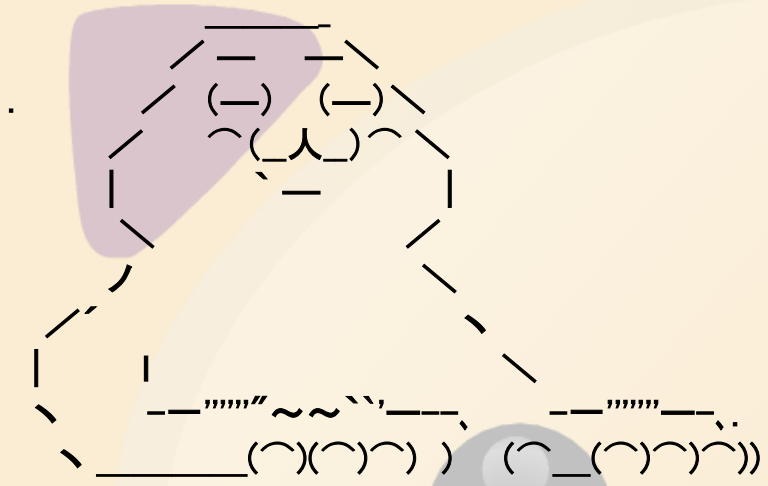
株式会社シーソフト代表取締役です。

現役のエンジニアです。プログラム書いてます。

にこにこカレンダーシートを販売しています。
2ちゃんねらーではありません。

Special thanks for 2ch.

前回までのおさらい



クラスに「時間軸」を考える。

時間軸に対するイベントが状態を作る

ステート／オブザーバパターン、DI

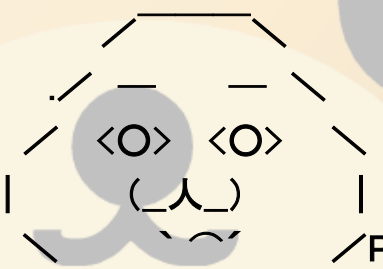
クラスの依存性を少なくし独立性を高めよう

本日のテーマ

変数

言語は何を使ってますか？

Ada Pascal PHP APL
Scala NC Shakespeare Whitespace
C++ C Smalltalk
Java C++ C++/CLI UML Objective-C
PostScript D言語
LISP PL/I アセンブラ なでしこ
ActionScript F# Tcl
SQL Ruby Visual Basic Mind
Delphi
Prolog Grass FORTRAN
COBOL Modula-2
LOGO Scheme Perl
ひまわり JavaScript
Eiffel ALGOL Python
R言語



プログラミング言語って覚えるの大変!?

制御文

if, for, while, switch, goto, return, (), {} ...

演算子

+, -, /, *, %, mod, and, or, xor, sizeof ...

データ型と変数

int, float, string, struct/class, variant, ...

ややこしいのはライブラリや統合環境

変数の種類

- グローバル変数
- 静的変数
- 動的変数
- ローカル変数
- 関数やメソッドの引数

- メンバ変数

グローバル変数

- 基本、つかっちゃだめ！

DLL内の共有変数で使ってしまいます
orz

静的変数

- 固定的な値などに利用される

```
static doube PI = 3.14159265358979323846;

printf( “Hellow C world¥n” );
```

メモリアドレス	メモリの内容
PI	PIの値
ある番地	Hellow C ...¥0

コンパイラにだけわかる

値の変化する変数を静的に配置すると、制御系システムやマルチスレッドシステムで利用するときに注意が必要。

動的変数

- ヒープなどのメモリを動的に利用する変数

```
int *Array = new int[ 100 ];

Person *man = new Person();
```

メモリのアドレス	メモリの内容
Array	A
man	B
A:ヒープ内	sizeof(int)×100
:	
B:ヒープ内	sizeof(Person)
:	

ローカル変数

- （一般に）スタックメモリを利用してモジュールの寿命だけ生成される変数

```
void foo()
{
   long ticks = GetTickCount();
   printf( “%ld¥n”, ticks );
}
```

便利。
多用に注意。

スタックアドレス	スタックの内容
	:
	“%ld¥n”番地
↑スタックポインタ	ticksの値
	fooの戻り番地
ticks	sizeof(long)
	スタックの保存
	fooの呼出元

関数やメソッドの引数

- ローカル変数と同様にスタックを利用する
  （ことが多い）

```
void foo()
{
   int sum = Add( 1, 2 );
   printf( “%d¥n”, sum );
}
int Add( int A, injt B)
{
    return A + B;
}
```

スタックアドレス	スタックの内容
↑スタックポインタ	:
	スタック保存
A	1
B	2
	fooへの戻り番地
	スタックの保存
	fooの呼出元

メンバ「変数」

- クラスや構造体のメンバ**「変数」**

```
class Person
{
    string Name;
    DateTime Birthday;
};

Person *man = new Person();
```

定義

宣言

定義ではメモリを消費しない

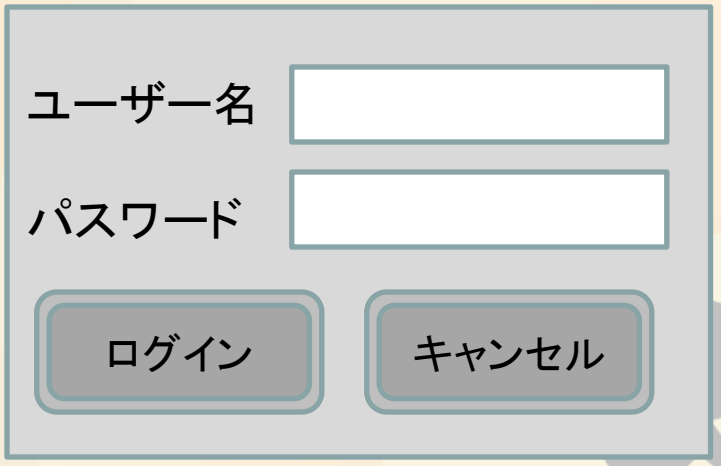
メモリのアドレス	メモリの内容
man	B
B:ヒープ内	sizeof(Person)
:	

ココに Name や Birthday が配置される
（この場合動的変数）

と、まぁココまでは、基本的な話ね。

実際にどのように使われているかな？

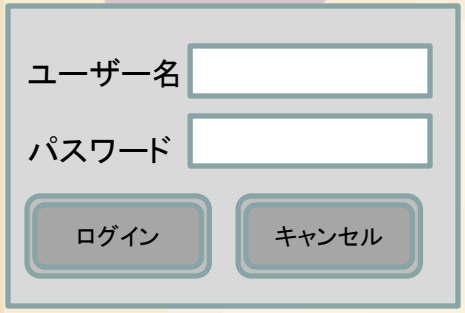
たとえば、こんな画面があったとしましょう。



ユーザー名とパスワードを入力
ログインボタンが押されたら、
入力情報が正しい文字の組み合わせになっているか検査
ユーザーが存在し、パスワードが正しいかを検証

どのように実装しますか？

ビューとロジックは、分けるよね？



ビュー

```
Form::Form()
{
}

Form::OnLoginClick()
{
}
```

ロジック

```
Logic::Logic()
{
}

Logic::Check (...)
{
}

Logic::Login(...)
{
}
```

データアクセス

```
DAC::DAC()
{
}

DAC::Connect(...)
{
}

DAC::QueryUser(...)
{
}
```

Formのコードイメージ

```
Form::OnLoginClick()
{
    string UserName = Text1->Text;
    string Password  = Text2->Text;

    bool ret = Logic->Check( UserName, Password );
    if( ret == true )
    {
        ret = Logic->Login( UserName, Password );
    }
    if( ret == false )
    {
            :
            :
```

編集テキストを内部で取得
ロジックに問い合わせ

Logicのコードイメージ

```
bool Logic::Check( string UserName, string Password )
{
    // UserName の妥当性検証
    // Password の妥当性検証
    return 真偽;
}

bool Logic::Login( string UserName, string Password )
{
    bool exist = DAL->QueryUser( UserName, Password );
    return exist;
}
```

DALのコードイメージ

```
DAL::QueryUser( string UserName, string Password )
{
    string SQL="SELECT COUNT * from UserTable "
                    "where (UserName=¥'%s¥')"
                    "and (Password=¥'%s¥')";
    try
    {
        SQL.FormatString( UserName, Password );

        DataBase->Query( SQL );
        if( DataSet->Count >= 1 )
            return true;
        }
    }catch( ... ){
    }
   return false;
}
```

【余談】保守性を下げる好き勝手な変数の命名

```
bool Logic::Login( string sUser, string sPsw )
{
    bool exist = DAL->QueryUser( sUser, sPsw );
    return exist;
}
```

アナタ以外の人が見ることを忘れずに

```
node *search(node *lhs, node *rhs);
```

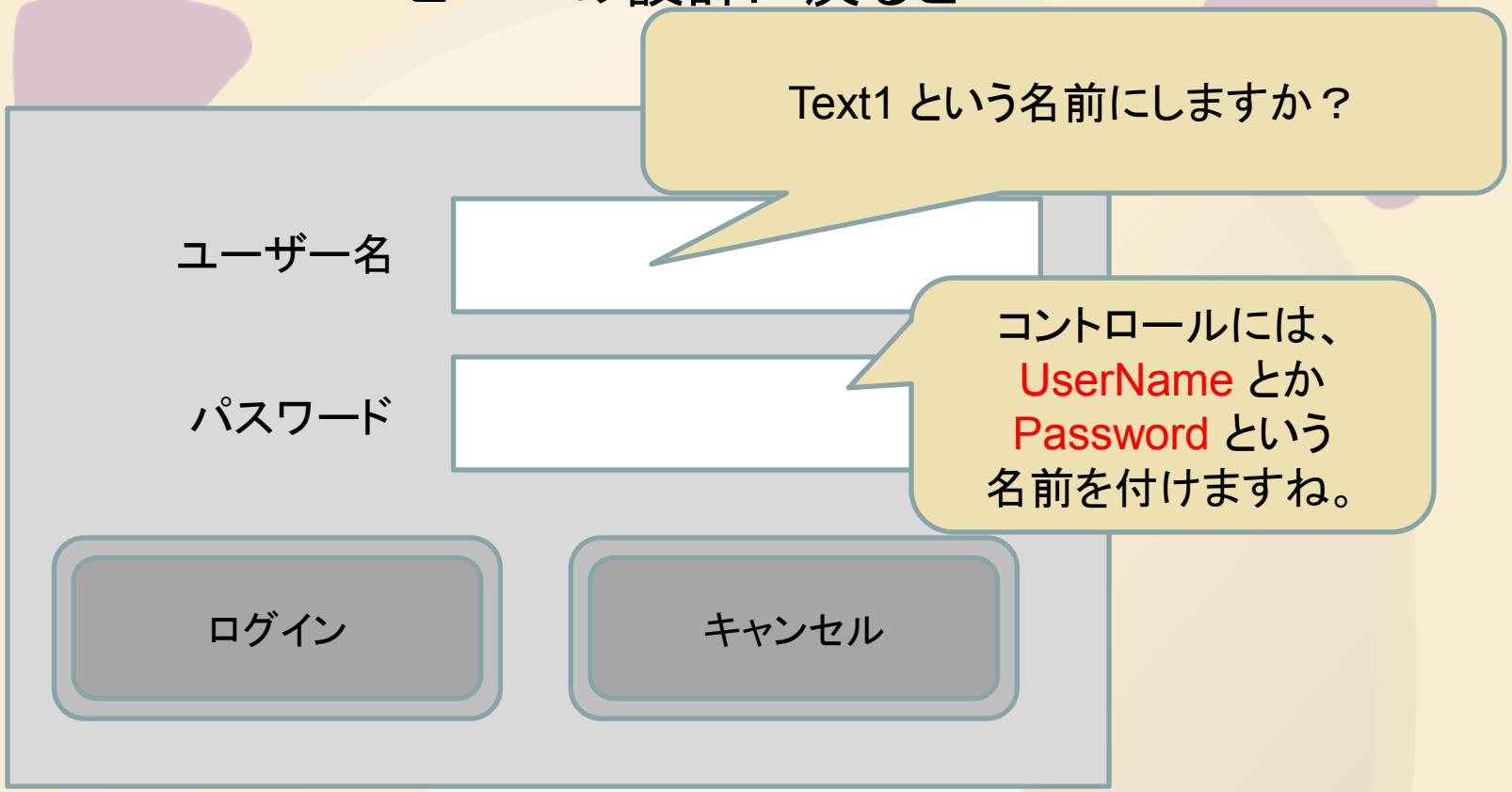
node *search(node *left, node *right);

統一された命名規則であることが重要！

ついでに、UserTableのイメージ

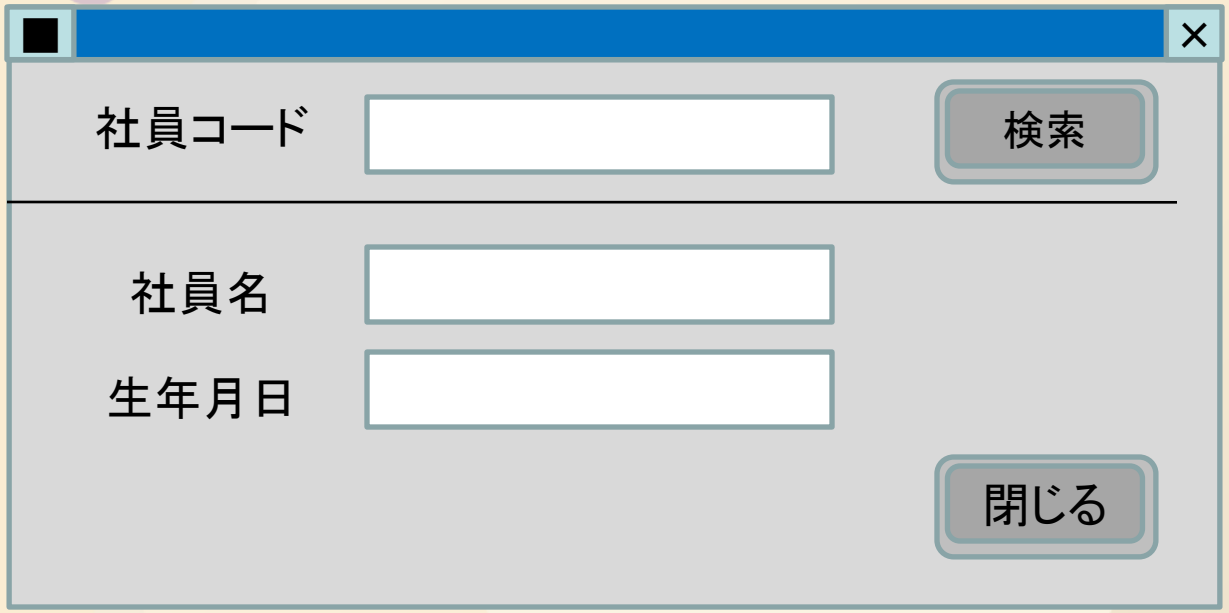
フィールド	型	サイズ	NULL許容
UserName	CHAR	40	×
Password	CHAR	20	×
:	:	:	:

ビューの設計に戻ると・・・

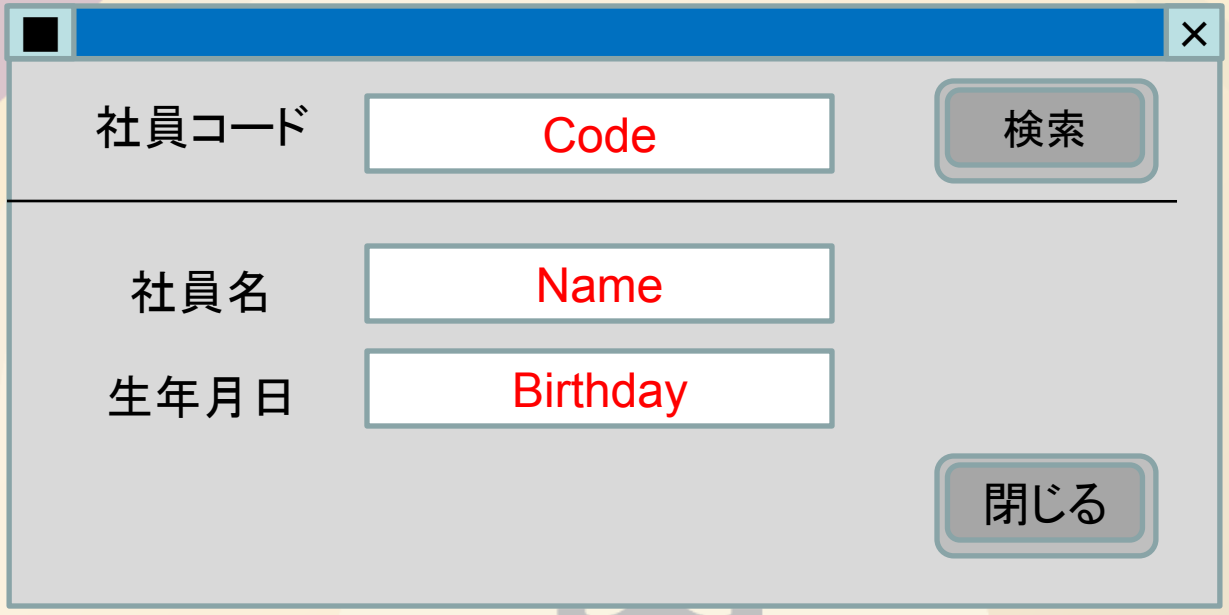

Text1 という名前にしますか？
ユーザー名
パスワード
コントロールには、
UserName とか
Password という
名前を付けますね。
ログイン
キャンセル

別の例を考えてみましょう。

こんな画面があったら？



画面の設計イメージ



Formのコードイメージ

```
Form::OnSearchClick()
{
    string Code = Code->Text;
    string Name, Birthday;

    bool ret = Logic->Check( Code );
    if( ret == true )
    {
        ret = Logic->GetPerson( Code, &Name, &Birthday );
    }
    Name->Text = Name;
    Birthday->Text = Birthday;
          :
```

Logicのコードイメージ

```
bool Logic::Check( string Code )
{
    // Codeの妥当性検証
    return 真偽;
}

bool Logic::GetPerson( string Code, string *Name, string *Birthday )
{
    bool exist = DAL->QueryPerson(Code, Name, Birthday );
    return exist;
}
```

DALのコードイメージ

```
DAL::QueryPerson( string Code, string *Name, string *Birthday )
{
    string SQL="SELECT * from PersonTable "
                    "where (Code=¥'%s¥')";
    try
    {
        SQL.FormatString( Code );

        DataBase->Query( SQL );
        if( DataSet->Count >= 1 )
            *Name     = Dataset->GetField("Name");
            *Birthday = Dataset->GetField("Birthday");
             return true;
        }
    }catch( ... ){
    }
   return false;
}
```

ついでに、PersonTableのイメージ

フィールド	型	サイズ	NULL許容
Code	INTEGER	8	×
Name	CHAR	40	×
Birthday	DateTime	8	×

ついでに、帳票設計があるとしたら？

社員名簿

コード	氏名	誕生日
Code	Name	Birthday

このように・・・

変数の名前って、あちこちで出てきますね。

毎回毎回、同じ名前を書かなきゃいけないよね。

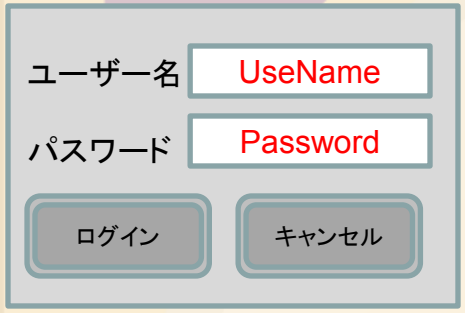
あたりまえ？

世界中のプログラマが、似たような画面やコードや帳票を毎日せっせと書いています。

めんどくさくないですか？

なんとかならないんでしょうか？

定義と宣言を見てみましょう。



ビュー

```
Form::Form()
{
}

Form::OnLoginClick()
{
}
```

コントロールとの入出力

ロジック

```
Logic::Logic()
{
}

Logic::Check (...)
{
}

Logic::Login(...)
{
}
```

Form／DALとの橋渡し

データアクセス

```
DAC::DAC()
{
}

DAC::Connect(...)
{
}

DAC::QueryUser(...)
{
}
```

Logicとのやりとり

フィールド	型	サイズ	NULL許容
UserName	CHAR	40	×
Password	CHAR	20	×
:	:	:	:

アプリケーションプログラム変数の宣言要因

- 画面のコントロール名
- データベースフィールド名
- 帳票ラベル／カラム名

そこで・・・

Form／Logic／DAL に設計情報から
自動的に変数を作り出してしまおう

という試み。

XMLファイルに定義情報を用意する

```
<Form Name="Form1">
  <UserInterface>
    <Field Name="UserName", DisplayName="ユーザー名", Type="String", ... >
    <Field Name="Password", DisplayName="パスワード", Type="String", ... >
  </UserInterface>
</Form>
```

```
<Database Name="AppDB">
  <Table Name="UserTable">
    <Field Name="UserName", DisplayName="ユーザー名", Type="String", ... >
    <Field Name="Password", DisplayName="パスワード", Type="String", ... >
  </Table>
</Database>
```

基本クラスでXMLを取り込み自動的に内部変数を用意する Form/Logic/DAL クラスを作る。

実は、一度作りました（笑）

.NETで考えてみよう。

WPFでXAML定義

DatabaseのXML

ReportのXML

基本クラス

現在、誠意開発中（笑）

ちょっとまって。ホントに便利になる？

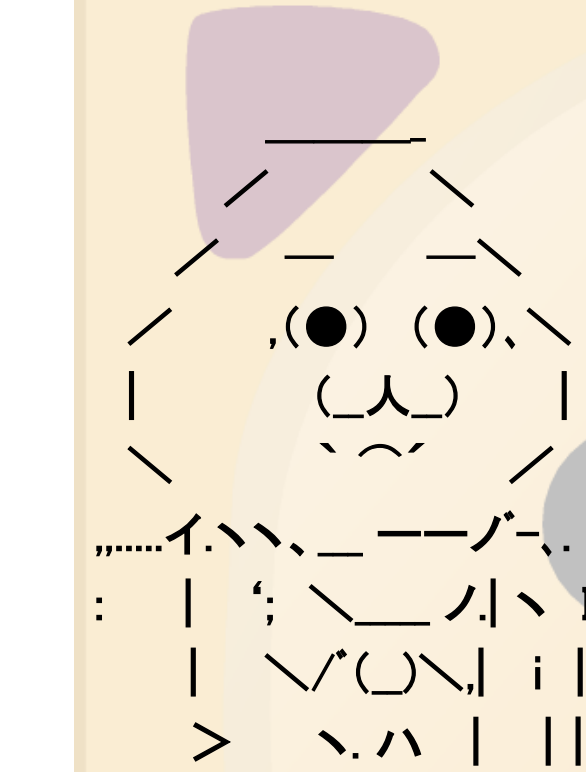
```
Form::OnSearchClick()
{
    string Code = Code->Text;
    string Name, Birthday;

    bool ret = Logic->Check( Code );
    if( ret == true )
    {
        ret = Logic->GetPerson( Code, &Name, &Birthday );
    }
    Name->Text = Name;
    Birthday->Text = Birthday;
          :
```

基底クラスの Fields[] 変数を利用してみると・・・

```
Form::OnSearchClick()
{
    bool ret = Logic->Check( Fields[“Code”] );
    if( ret == true )
    {
        ret = Logic->GetPerson( Fields[“Code”], Fields[“Name”],
Fields[“Birthday”] );
    }
    Name->Text = Fields[“Name”];
    Birthday->Text = Fields[“Birthday”] ;
          :
```

よけいにめんどくさいじゃんか！
（笑）


変数だけを略してもダメ！



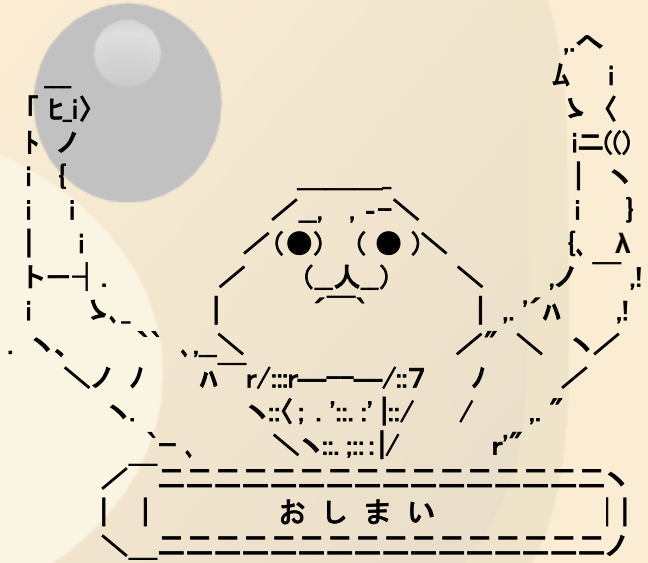
残念ながら時間がきてしまったようです。

次回は、自動的に作られた変数を参照しなくてすむ
新たな方法論に展開するでしょう。

実装がまったく間に合いそうにないけどw

ご静聴ありがとうございました。

m(_._)m



Special thanks for Yaruo charactors