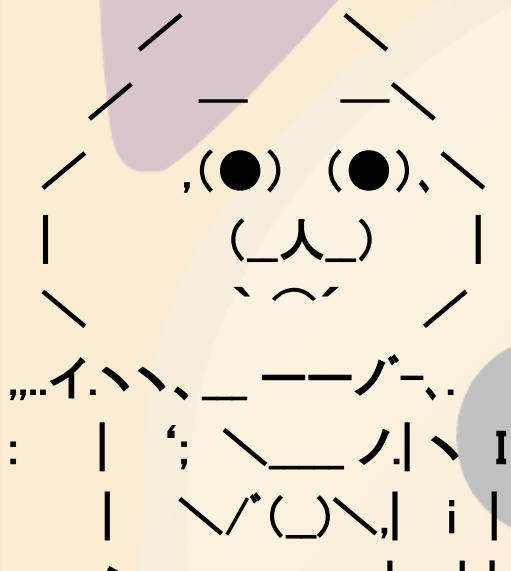
匠の伝承w

マルチな時代の設計と開発

パート6

スピーカー自己紹介



ゆーちです。

本名は、内山康広といいます。
31歳(H)です。おっさんです（笑）

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。
BkReplyer をシェアウェアにさせていただきました。

Special thanks for 2ch.

前回のおさらい

変数の基本的な話

変数を自動化してみようという話

たとえば、こんな画面

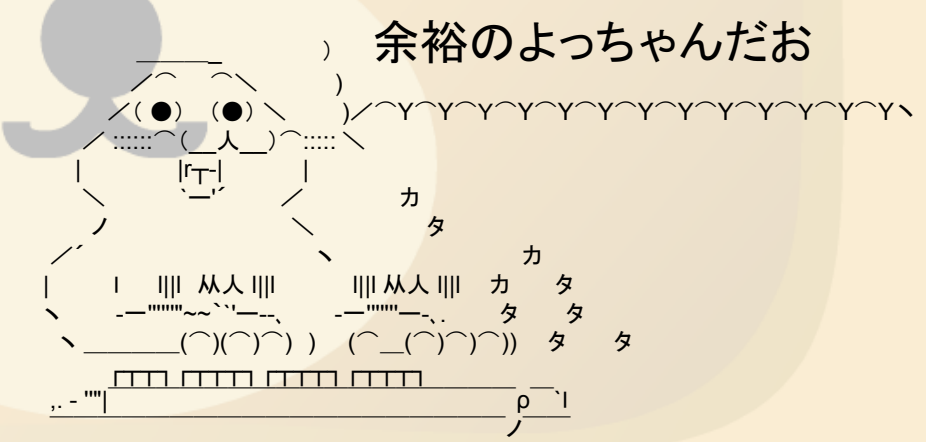
ユーザー名

パスワード

ログイン　キャンセル

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

余裕のよっちゃんだお



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

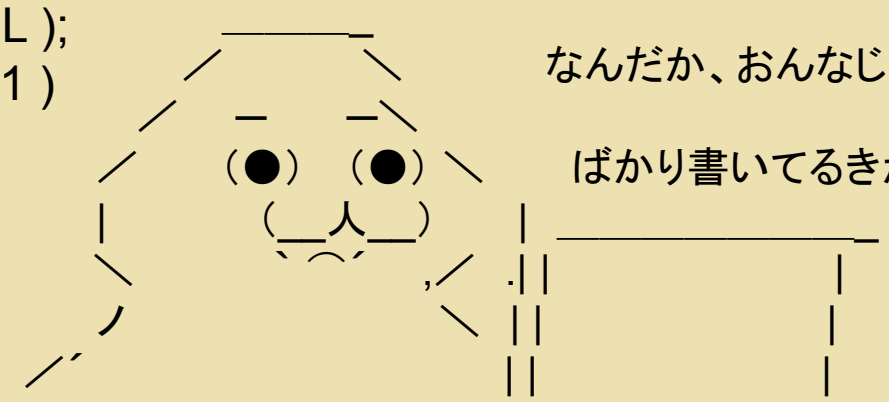
```
bool Logic::Check( string UserName, string Password )
{
    // UserName の妥当性検証
    // Password の妥当性検証
    return 真偽;
}

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;
}
```



なんだか、おんなじこと

ばかり書いてるきがするお

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。

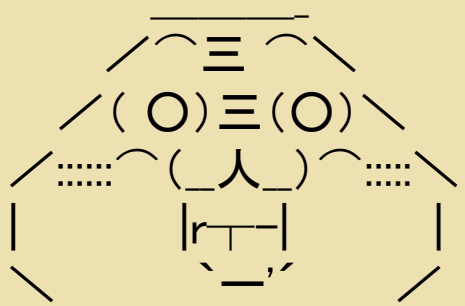
ちょっとまって。ホントに便利になる？

```
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 )
    {
            :
            :
```

基底クラスの Fields[] 変数を利用してみると・・・

```
Form::OnLoginClick()
{
    bool ret=Logic->Check(Fields[“UserName”], Fields[“Password”]);
    if( ret == true )
    {
        ret = Logic->Login(Fields[“UserName”], Fields[“Password”]);
    }
    if( ret == false )
    {
          :
          :
```



よけいにめんどくさくなったお

変数だけを略してもダメ！

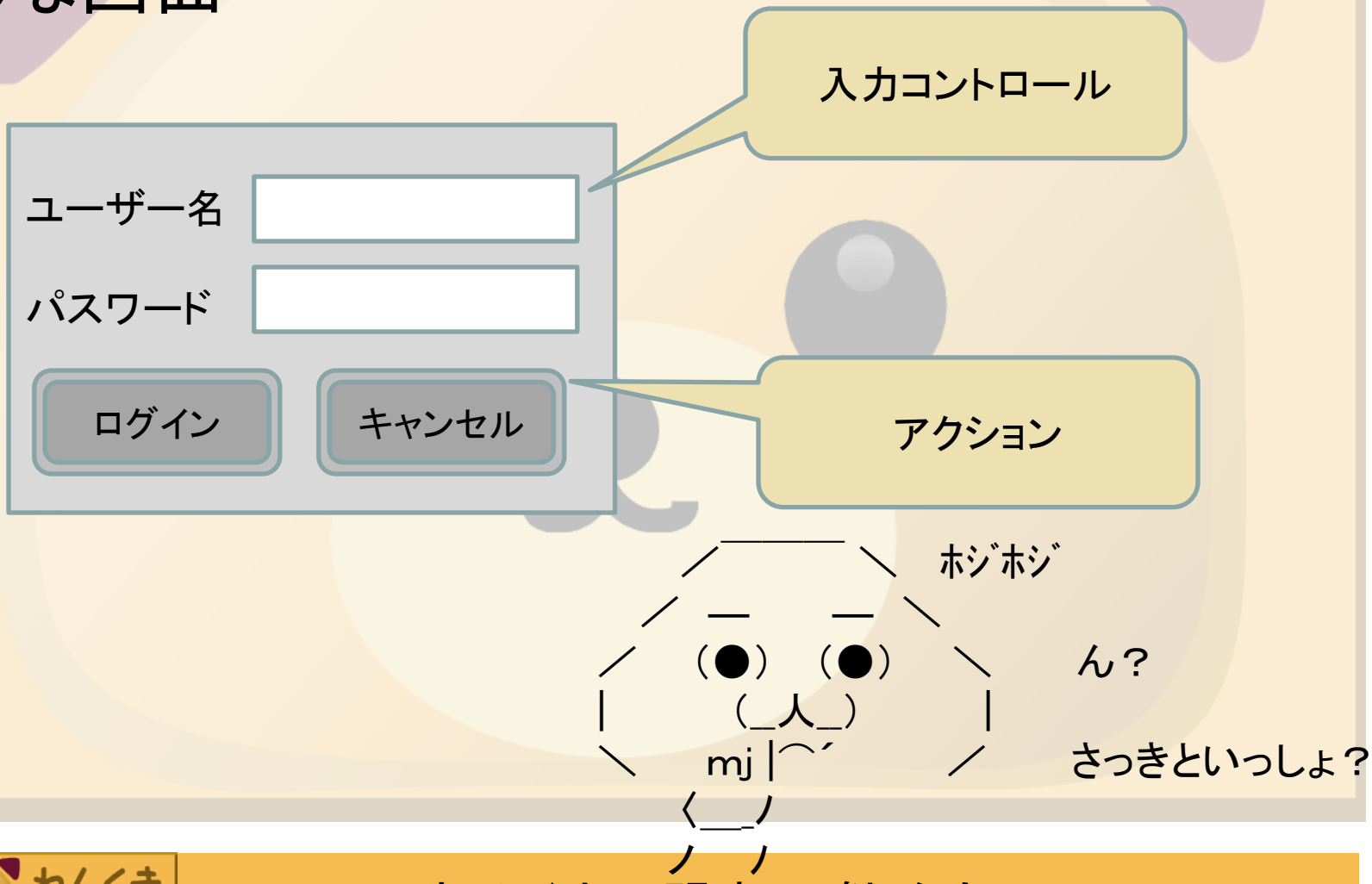
本日のテーマ

アプリケーション・パターン

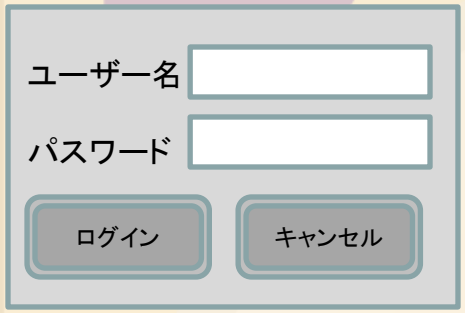
似通った処理をコンポーネントにしてしまおうぜっ！みたいな(笑)

ちょいと具体例を挙げていってみましょう

こんな画面



ビューとロジックの内部処理



ビュー

Form::OnLoginClick()
{
}
コントロールから値を取り出し、ロジックでCheck後にLogin

Form::OnCancelClick()
{
}
ロジックでCancel

ロジック

Logic::Check (...)
{
}
文字妥当性
登録されてるの？

Logic::Login(...)
{
}
ログインの記録とか

Logic::Cancel (...)
{
}
終了処理

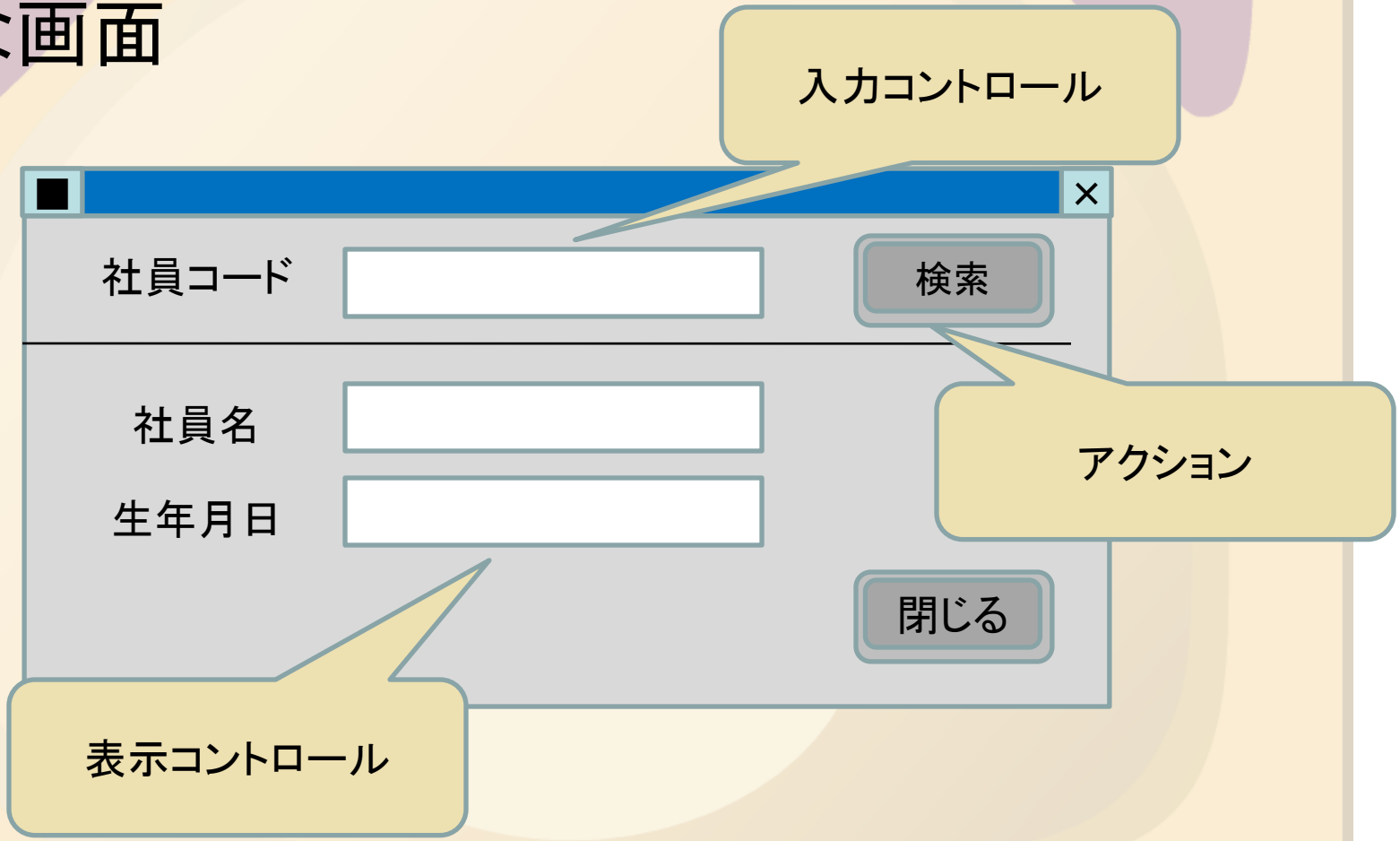
データアクセス

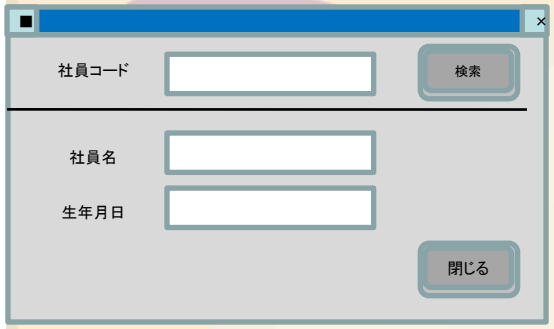
DAC::QueryUser(...)
{
}
登録されてるの？

DAC::Connect(...)
{
}
なにかする

別の例を考えてみましょう。

こんな画面





ビュー

Form::OnSearchClick()
{
}

コントロールから値を取り出し、ロジックでCheck後にSearch

Form::OnChangeField()
{
}

ロジックから値の変更通知を受け取ったら、コントロールに設定

ロジック

Logic::Check (...)
{
}

文字妥当性
登録されてるの？

Logic::Search(...)
{
}

社員コードでデータ検索

データセットからフォーム用変数の通知

データアクセス

DAC::QueryUser(...)
{
}

登録されてるの？

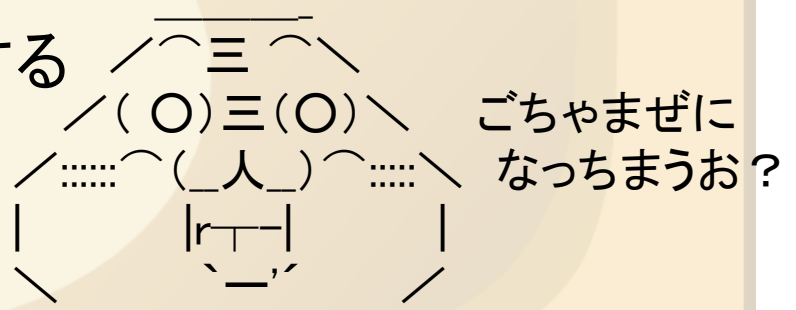
DAC::Search(...)
{
}

社員検索SQL発行

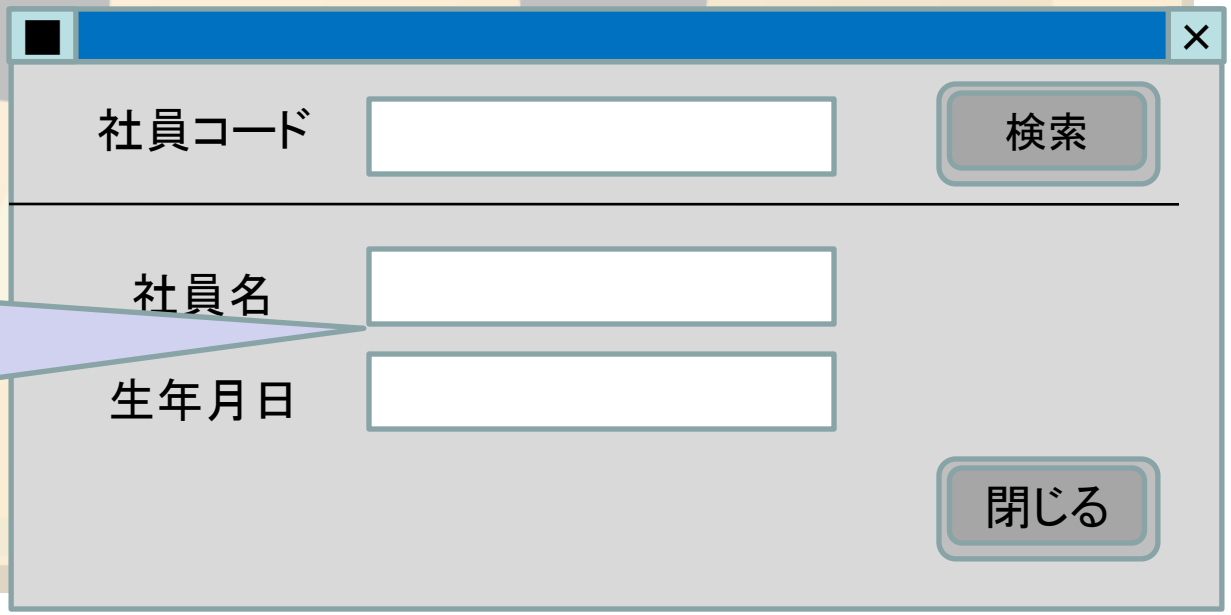
これら2つを1つにできないでしょうか？

```
BaseForm::OnActionExecute()
{
    //入力コントロールを列挙
    foreach( Control *control = InputControls  ){
      // ロジックのプロパティに入力値を転送
       Logic->Fields[control->Name] = Fields[ control->Name];
    }
     // ロジックに値チェックをお願いする
     if(   Logic->Check() ==   true ){
          Logic->Execute();
    }
```

注意：コードはあくまでもふいんきwです。



```
BaseForm::OnPropertyNotifyChanged()
{
    //表示コントロールを列挙
    foreach( Control *control = DisplayControls  ){
      // ロジックのプロパティに入力値を転送
       Fields[control->Name] = Logic->Fields[control->Name];
    }
    Repaint();
}
```



表示コントロールに
値が設定される

Logicはどうなる？

```
bool BaseLogic::Check()
{
    return true;
}
bool BaseLogic::Execute()
{
    return true;
}
```

検証すべきフィールドや条件は毎回異なるので、基底クラス(BaseLogic)では汎用的な実装はできませんね。

何を実行するのか、というのも要件によって異なりますので、基底クラスで実装できにくいですね・・・

（意味なさそうだお）

派生クラスで個別対応？

```
class LoginLogic : public BaseLogic …

bool LoginLogic::Check()
{
    return DAL->QueryUser(Fields[ “UserName” ],
                          Fields[ “Password” ]);
}
```

仮にロジックを個別実装することになったとしても、
この方法が実現できれば、少なくともフォームの
コードについては基底クラスが処理してくれるので、
画面設計だけやっておけばロジックの必要箇所の
実装だけですむことになります。

標準のアクションリスト

一般的な業務アプリケーションの場合、オペレータのアクションの入り口（メニューやボタン）には、次のようなアクションが用意されます。

「キャンセル」

「OK（実行）」「更新」「登録」「確定」

「削除」「検索」

「前ページ」「次ページ」

「コピー」「貼り付け」などなど

定型ロジックを基底クラスにうめこんじゃおう

Formの[OK]処理

入力フィールドを検証（Check）

＞チェックNGの場合、メッセージ表示

＞間違った箇所にカーソル移動

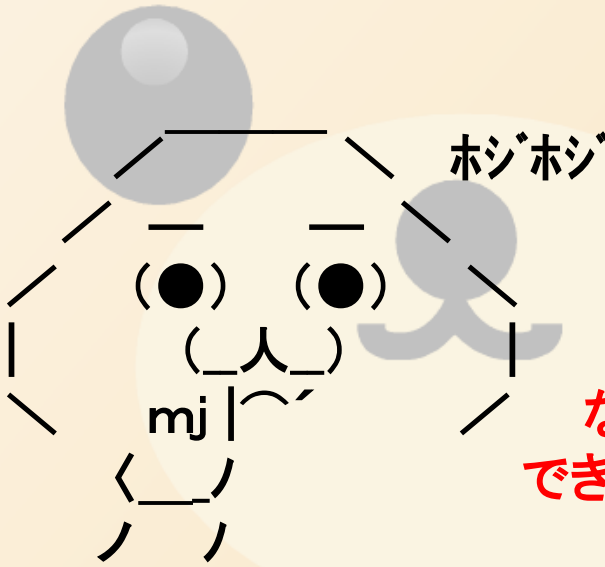
チェックOKなら、実行処理(Execute)

＞必要な保存処理を実施する

Formの[キャンセル]処理

フォームを閉じる。

アプリケーションフォームならプログラムを終了する



え一？？？
このくらいまでならわかるけど
なかなかそう簡単に共通化は
できないんじゃなぁい？？？

グリッドを持つ画面


月間集計表
月
Month
表示
商品コード
商品名
顧客コード
数量
単価
金額
ProductId
Product
CutomerId
Count
UnitPrice
Amount
赤い文字はXMLで定義された名前
前ページ
次ページ
閉じる

[表示]ボタン処理

基底データアクセスロジックの一部・・・

```
string SQL = “SELECT “
foreach( field = OutputFields ){
   SQL += field->Name;
   if( field != End of List ){
      SQL += “,“;
   }
}
SQL += “from “ +DatabaseTable;
SQL += “WHERE “;
foreach( field = InputFields ){
    SQL += InputFields->Name + “= ¥””
           + InputFiels->Value + “¥””;
    if( field != End of List ){
        SQL += “&&”;
    }
}
```

SELECT
ProductId,Product,CutomerId,Count,UnitPrice,Amount

from UserDatabase.**UserTable**

where
Month = “5”;

この辺もXMLで定義

ページ行数＋1を選択とかもあり。

[次ページ]ボタン処理（あくまでもイメージw）

```
BaseLogic::NextPage()
{
   DAL->NextPage( CurrentPage );

   foreach( Grid[ “グリッドの名前” ].Fields )
   {
      Grid[グリッド名].Field[ フィールド名 ] = DAL->Dataset[ フィールド名 ];
   }

   Observers->Notify( GRID_CHANGE, グリッド名);
}
```

SQL発行

データセットからローカルに値を取得

グリッドを表示しているフォームに変更通知

データベースがあるとは限らない！

そのとおり。

DBとは異なる格納なら、DALだけかえてしまえばいいんじゃない？

（設定情報をiniファイルに書いたりレジストリにしたり）

データベース、ini ファイル、レジストリの入出力があれば、多くのアプリケーションでは満足なのでは？

通信を利用するアプリケーションに適用できない！

そんなことはないっす。

一般的な通信手段として

・TCP/IPとか

・COMポートとか

は、標準的なやりとりで吸収できます。

クラサバ

- サーバーアプリケーション

  ServerSocket

- クライアントアプリケーション

  ClientSocket

たとえば、どうなる？

典型的な例では、「マスターメンテナンス画面」

テーブル設計
メンテナンス画面
それぞれのXML定義

ほんとかお？

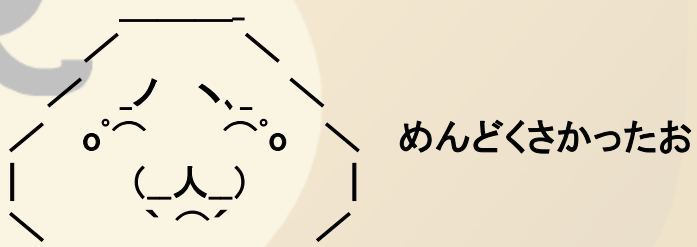
よろしくおながいしますお！

だけで、コードを書かずにできあがってしまう！

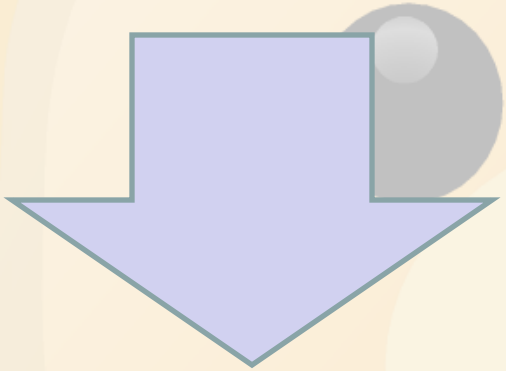
たとえば、どうなる？

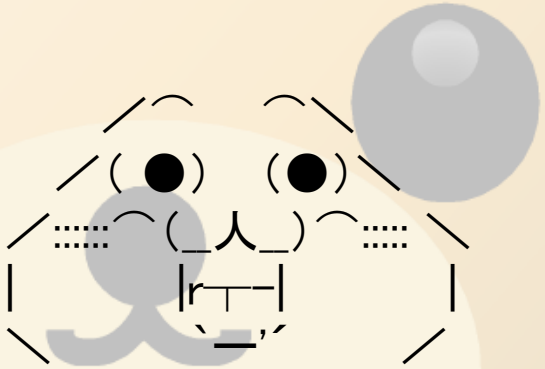
アプリケーションの設定情報

画面レイアウト、XML定義だけでコードを書かなくて、できちゃう。



プログラムの開発時間を大幅に短縮できる！



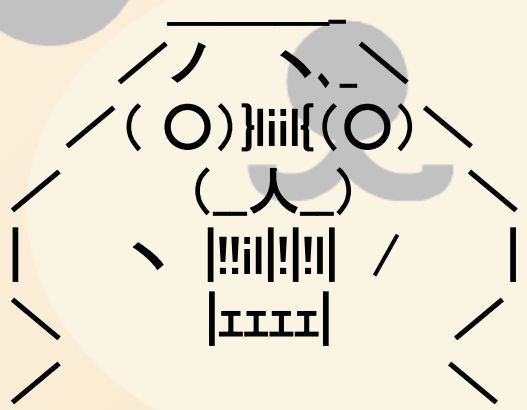


ほんとかお？

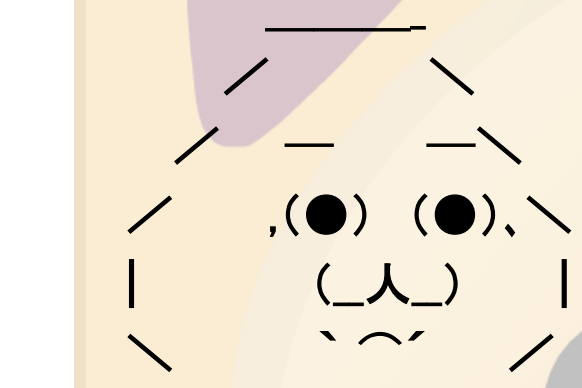
定時で帰っていいのかお？

デスマーチからの解放！

現在、誠意開発中（笑）



まだできてないのかお！



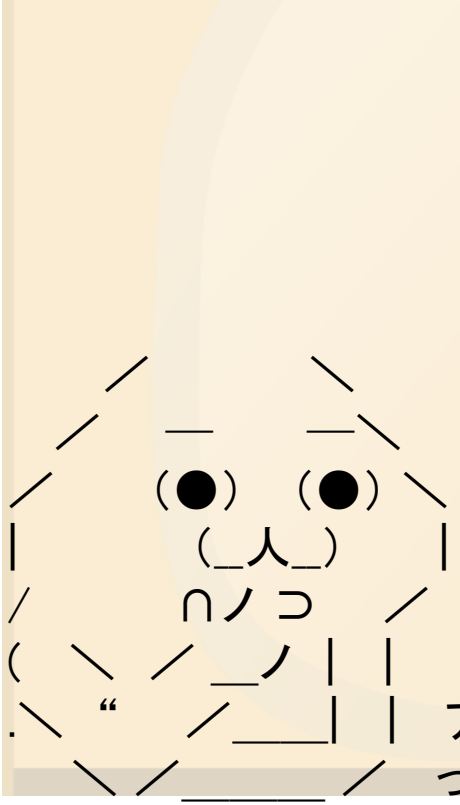
残念ながら時間がきてしまったようです。

次回は、ここで紹介したフレームワークを
実際にデモを交えてご紹介しましょう。

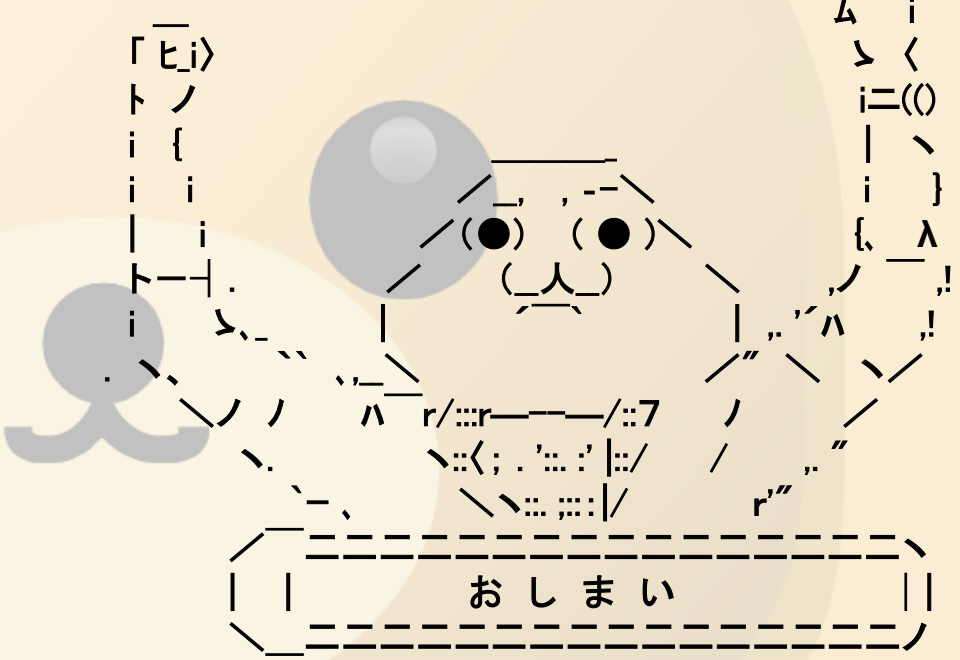
いつになるのか、皆目見当もつかないわけだw

ご静聴ありがとうございました。

m(_._)m



うーん・・・



アプリケーション・パターン
ってなんだったのかお！？

Special thanks for Yaruo charactors