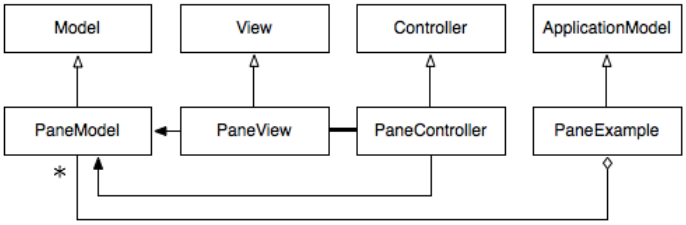
■ 2011/4/6

「使わないと損をする Model-Controller-View 」を下敷きに。

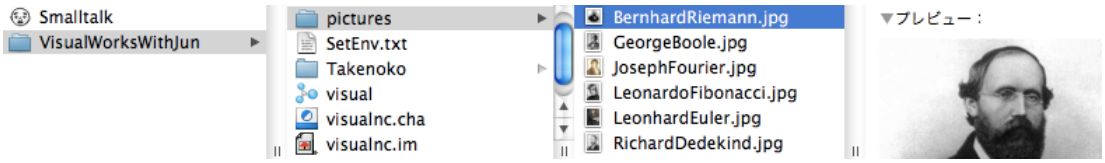
今回作りたいものはこんなもの。



今回は新年度のため操作説明を手厚めにやった。ので操作に慣れた人は最後の「まとめ」まですっ飛ばすが良し。（安田）

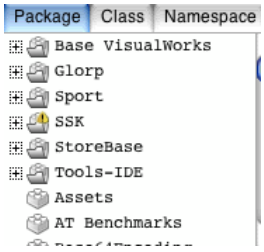
□ 準備

配布された画像データが入っている pictures フォルダを VisualWorksForJun フォルダの直下（VisualWorks アプリのある隣）に置く。



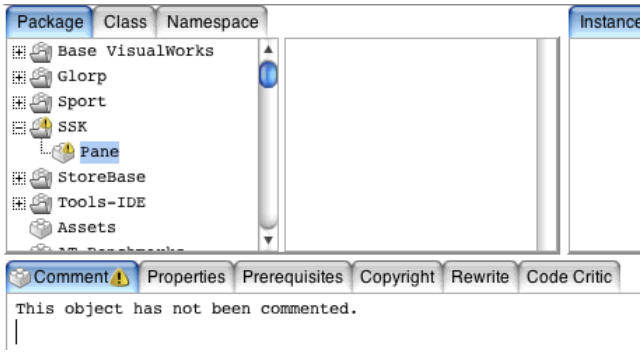
System Browser をあけてPackage >> New Bundle... でバンドル（パッケージをまとめているもの）を作る。

Parent bundle の指定によって場所（階層的な位置）を指定できるが、今回は上記のように空でやろう。
親無し、なので、一番上に出来る。



この SSK Bundle を選択した状態で Packages >> New Packages... メニューを実行。Pane という名前のパッケージを作る。

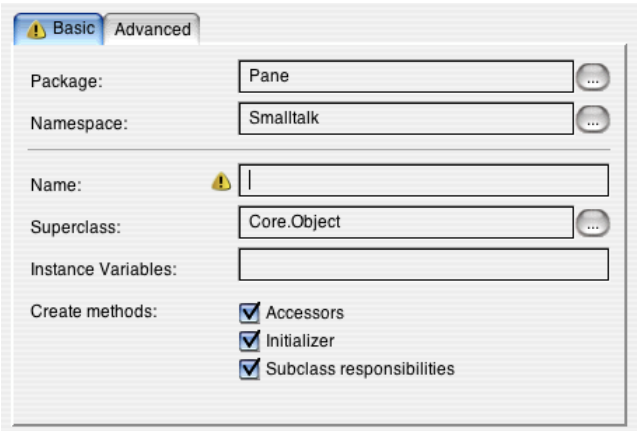
できた。
下に示すようにビックリマークがついているが、これはコメントがない、ということ。



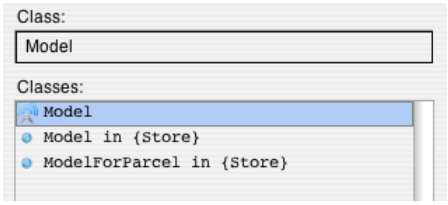
ココをとりあえず入れる。ビックリが SSK bundle, Pane package 両方についていることから判るが、
SSK Bundle と Pane Package 両方とも適当に何かコメントを入れて、コンテキストメニューで accept しておく。
するとビックリマークは消える。

□ クラスを作る（まずは PaneModel ）

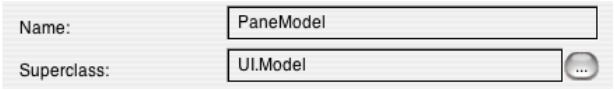
作るクラスは先に提示したクラス図どおり、PaneModel, PaneView, PaneController, PaneExample の四つ。
Pane パッケージを選択した状態で、Browser の Class メニューから New Class... を実行。
まず PaneModel を作るか。最初はクラス名を入れる。



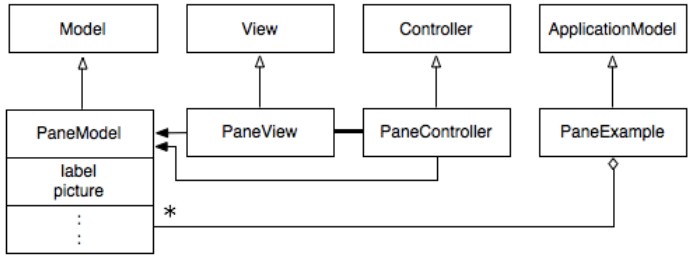
クラス名 PaneModel として、Superclass を変更する。つまり Core.Object ルートオブジェクトのサブクラスではなく、Model クラスのサブクラスとする。そのために Subclass: フィールド横の  ボタンをクリックして下の Select ダイアログを出す。



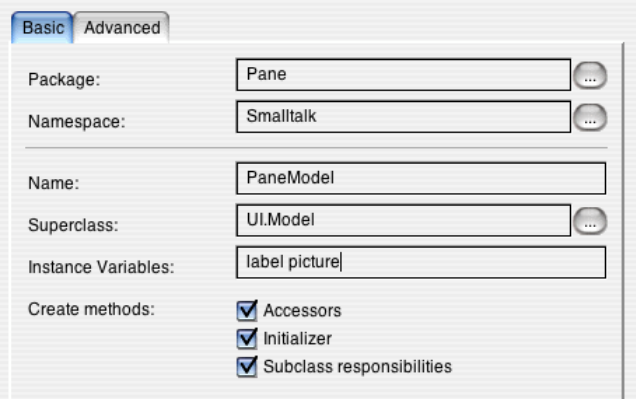
ここで Model と入力すると候補が挙がるが、ここで Model （ブレース無し）を選んで OK を押す。
（つまり標準的に Smalltalk で Model といったら MVC の Model なのだな）
結果、先の New Class... ダイアログの設定は以下のようになる。



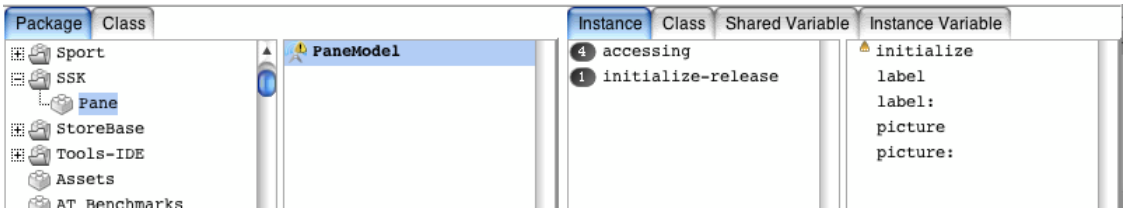
更にインスタンス変数として label, picture というのを想定しておく。
つまりこういうのを作るんだ、と。



で、それをまたこのダイアログに指定しておく。つまり最終的には以下のようになる。

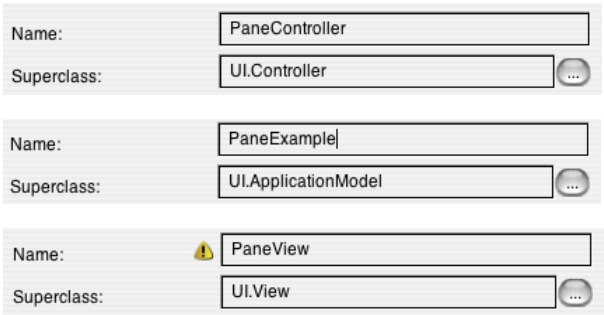


Accessors や Initializer のチェックを入れているので、この状態で OK とすると、label, picture に対するセッター、ゲッターが作られる。
出来上がりを確認するとこうなっている。Paneクラスができただけでなく、インスタンス側のプロトコル（メソッドカテゴリ）と initializer, setter, getter 相当のメソッドが自動的に作られている。



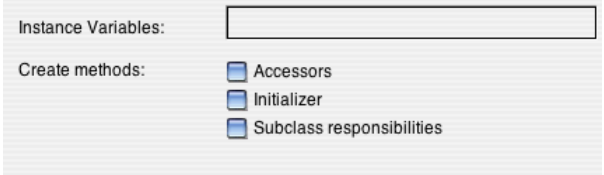
□ 残りのクラス

PaneController, PaneView, PaneExample を作る。
同様に Class >> New Class... メニューを選択し、Name と SuperClass を指定する。
それぞれこのようになる。



（PaneView のコーションマークは安田がヘンな操作をしたせいで普通は出ない）

これらのクラスについてはインスタンス変数でこの時点で予定されているものはないので、それらについて空欄、チェックなしのまま。



とりあえずこれでOK。以下のようにできあがった。

□ 保存

ここで一旦保存。
SSK Bundle を選んだ状態で Packages >> File out >> Package... を選択。

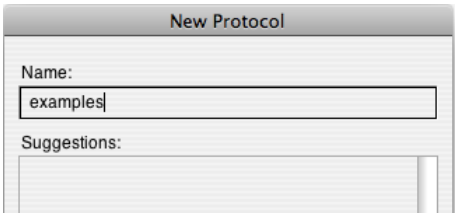
（SSK Bundle を選んでコンテキストメニューでも可）

□ メソッドの実装

まずは PaneExample の example メソッドを作っておく。
（非常にトップダウンに作っているのが面白い。Smalltalk ではクラスを見たらまず最初に Example を見ろ、ということらしいのだけれど、つまり作っている時からそこが入り口になってるわけね。）

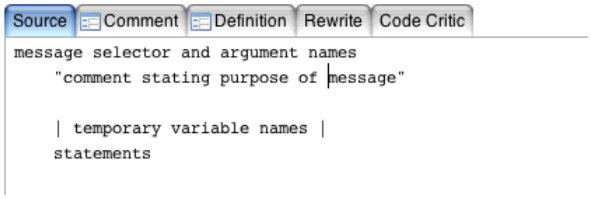
PaneExample のクラスを選んで、クラス側のタブを指定。プロトコル欄の instance creation を選択して右クリック。

ここで New.. を選んで、examples プロトコル（メッセージカテゴリ）を作る。
（これ、つまり example プロトコルを作りたいだけなので、Browser の Protocol >> New... メニューでもいいんだろうなあ。）



名前を examples と指定して、OK とするとこんな感じ。

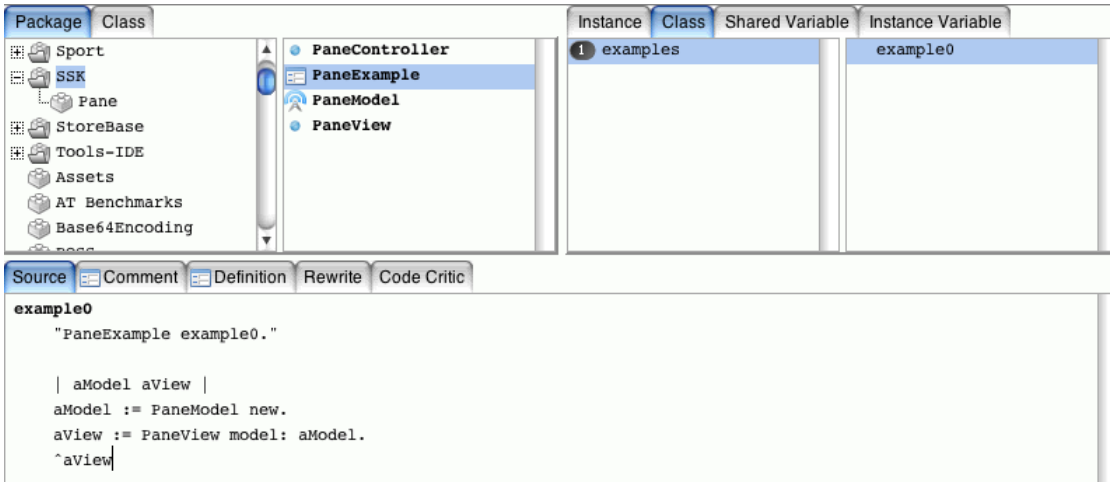
できたがプロトコルだけで中身は空（なのでカウントもゼロ）。
ここでそのまま下のペインに示されている Source のテンプレート（下図）に、そのまま新しいメソッドを書き込む。



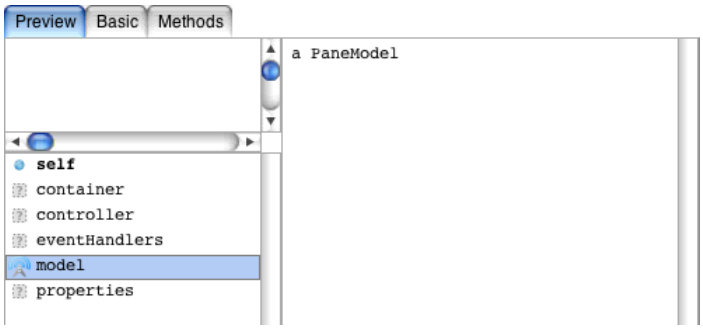
このテンプレートに従って、

example0
"PaneExample example0."
| aModel aView |
aModel := PaneModel new.
aView := PaneView model: aModel.
^aView

と入れて、format して accept。



試しに PaneExample example0 を Do it してみて、、も何も判らないので Inspect it する。（これで動くというのも面白いが）



model を見るとちゃんと aPaneModel が入ってる。
PaneView クラスに model: メソッドなんて作ってないのに動くのは上位クラス（View）にあるということだわねえ。

次にこの PaneView のインスタンスに controller をつけてみよう。とりあえず。

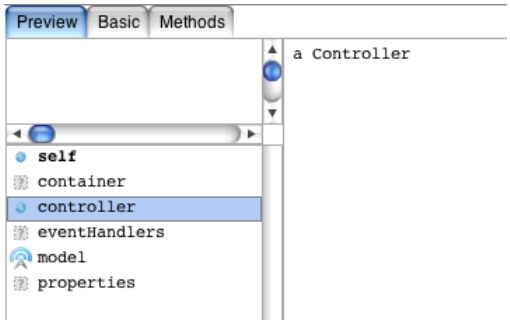
example0

```
"PaneExample example0."
| aModel aView |
aModel := PaneModel new.
aView := PaneView model: aModel.
aController := aView controller.
^aView
```

aController は要らない。

```
aView controller.
```

でもいい。これでコントローラが aView （PaneView のインスタンス）につくはず。



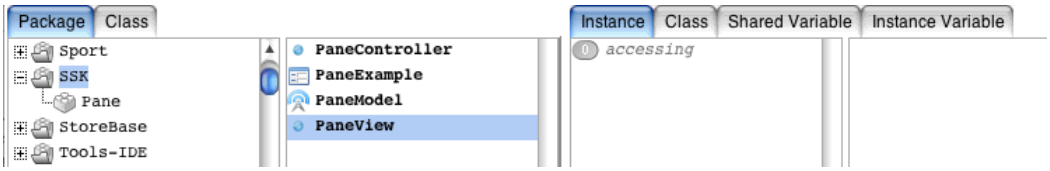
ついた、、、が、よろしくない。つまり上位クラスは Controller のインスタンスではなく PaneCotroller のインスタンスになってほしい。
ので書き換える。（赤字部分）

example0

```
"PaneExample example0."
| aModel aView |
aModel := PaneModel new.
aView := PaneView model: aModel.
aView controller: PaneController new.
^aView
```

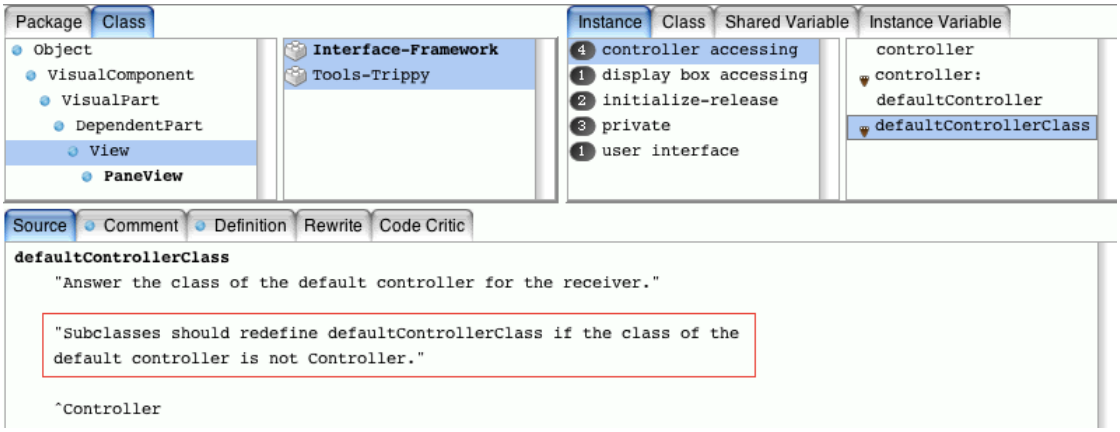
が、これ、超カッこわるいんだそうで。
　aView := PaneView model: aModel.
と書くべきだ、というのが青木さんの主張。

ちょっとチェック。下のような状態にする。



つまりPaneView のインスタンス側を見る状態にして、ここで Class 階層構造を見る（Packages タブのすぐ隣の Class タブをクリック）

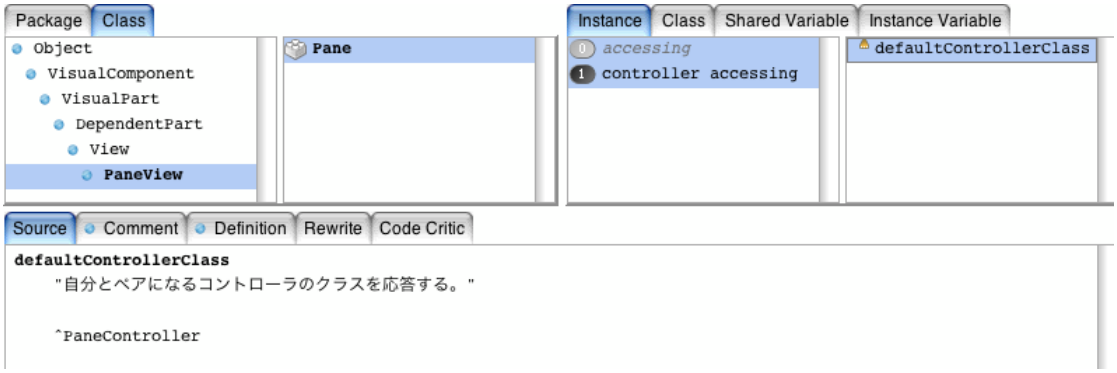
この状態で階層一つ上の View を見る。そこのインスタンス側の controller accessing カテゴリを見る。



ここの赤枠部分に注目。「ここのサブクラスで再定義すべき」とある。
というわけでこの部分をカスタマイズしたいので、このメソッドをまずは真似っこして、自分の PaneView に放り込む。（独自に実装する）

defaultControllerClass
　"自分とペアになるコントローラのクラスを応答する。"
　^PaneController

こんなのでどうか。これで accept しておく。



＃この時点で気がついたが、青木さんは accessing プロトコルを remove した模様。
＃というか Protocol メニューで New... せずRename... したか。まあいいや。

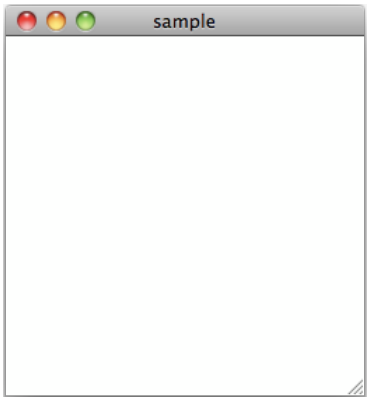
こうする限りは、PaneExample class の example クラスメソッドについては、
　aView controller: PaneController new.
としないで、
　aView controller.
で良いことになる。

＃new が上位クラスで書かれていることを確認してないけど、ま、いいか。また今度。

□ ウィンドウをあける

Example メソッドの後半を更にいじる。
せっかく作ったさっきの aView controller を含めて、それ以降を書き直し。

example0
```
"PaneExample example0."
| aModel aView aWindow |
aModel := PaneModel new.
aView := PaneView model: aModel.
aWindow := ScheduledWindow new.
aWindow component: aView.
aWindow open.
^aWindow
```
実行するとものすごく小さいウィンドウが開く。そこで
```
aWindow openWithExtent: 256 @ 256.
```
と書き換える、と思ったがこう書き換えると何かしらおかしな挙動をする。
僕だけ？その後濱崎さんが見たら（何もせず視線を投げるだけで）動いた。ま、いいや。
ウィンドウラベルも入れたいので、下記も追加。
```
aWindow label: 'sample'.
```
これで実行するとこうなる。

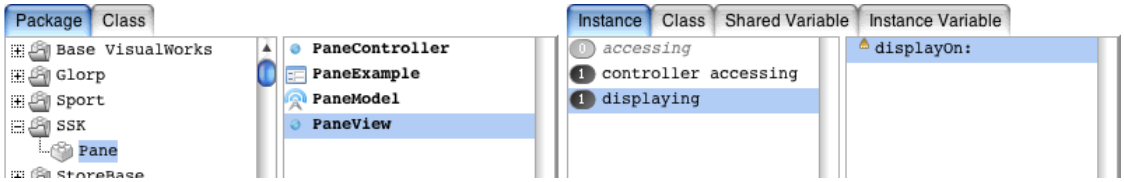


この時点の出来上がりコード。
example0
```
"PaneExample example0."
| aModel aView aWindow |
aModel := PaneModel new.
aView := PaneView model: aModel.
aWindow := ScheduledWindow new.
aWindow label: 'sample'.
aWindow component: aView.
aWindow openWithExtent: 256 @ 256.
^aWindow
```

そろそろ次のステップのために example1 メソッドにこの内容をコピーしておく。

□ 窓 MVC を作る

PaneView の instance のプロトコルとして displaying というのを作り、インスタンスメソッドとして、displayOn: を作る。



とりあえず今はウィンドウ上に描画するのは難しいので Transcript で何か出して動作していることを確認する。

displayOn: graphicsContext
```
"自分自身を表示する。"
Transcript
    cr;
    show: thisContext printString;
    space;
```

show: Time now printString

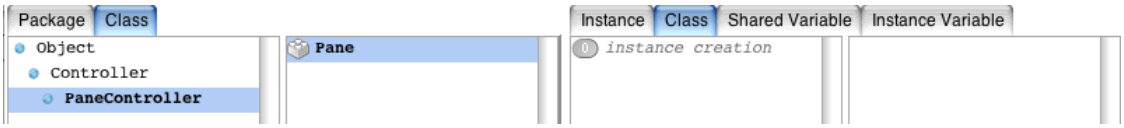
これで example1 でも実行すると、Transcript にこのように出る。

PaneView>>displayOn: 20:21:47

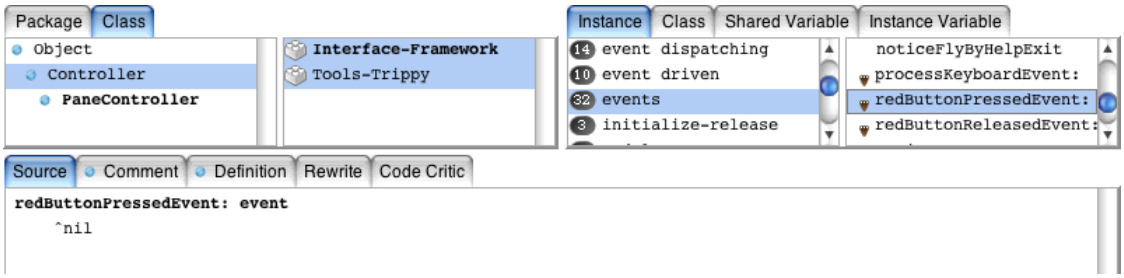
ウィンドウのサイズを変えたりする度に呼ばれるのが判る。なるほど画面（再）描画要請なんだな。

□ Controller

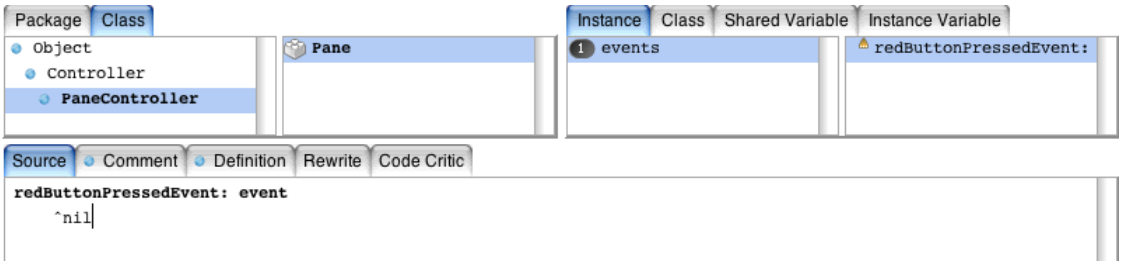
再び Controller のクラス側を選んで、クラス階層表示にする。



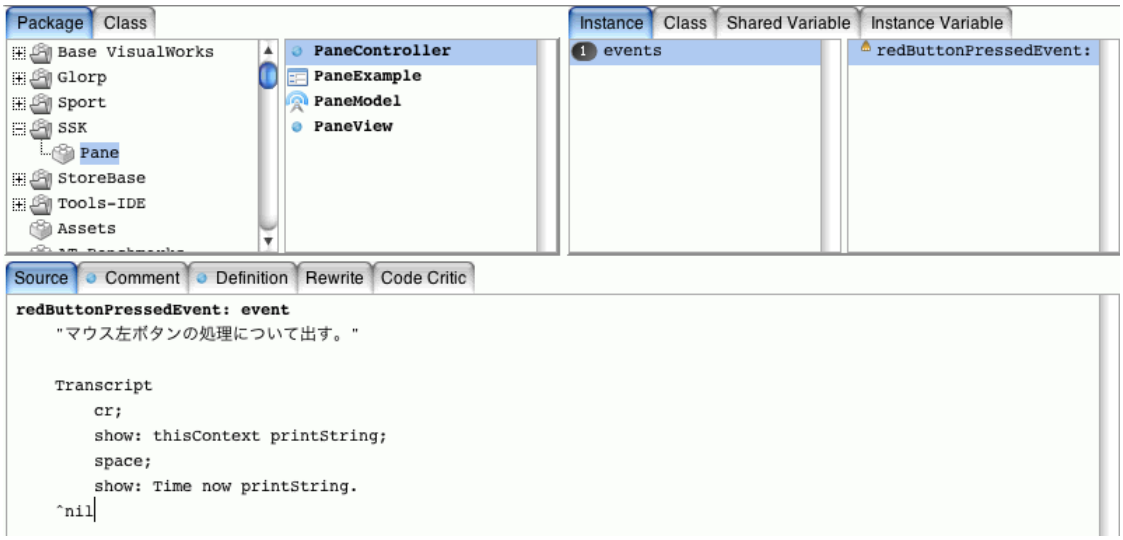
これで上位のController にある、ボタンクリックに反応するメソッド、redButtonPressedEvent: メッセージを出す。カテゴリ events にある。



実装はスケルトン状態で空（nil を返すだけ）なのね。つまりここをサブクラス側で実装すれば良いわけだ。
再びこのソース部分をコピーして、下位の PaneController のインスタンス側 events カテゴリを選択して、ソース部分に貼り付けて accept する。



で、ここにさっきの Transcript 出力のコードをもってきておくか。



この状態で（アプリを立ち上げ直す必要無し）マウスを（左ボタンで）クリックすると Transcript ウィンドウに以下のように出る。（赤文字が増えた部分）

PaneView>>displayOn: 20:23:14
PaneController>>redButtonPressedEvent: 20:28:52

ひひひ。

次、redButton を yellowButton に換えて accept 。つまり同じ events カテゴリに右ボタンクリックの動作を追加した。動作確認。

PaneView>>displayOn: 20:23:14
PaneController>>redButtonPressedEvent: 20:28:52
PaneController>>yellowButtonPressedEvent: 20:29:40

うひひひ。

他、いろいろホイールなども対応しているので上位クラス見て捜すと良し。

動作途中のシステムに直接的に手を入れているのが面白い。
（青木さんはこれを full reflective だ、と強調している。前にもやった。まあインタプリタだからなあ。。。LISP がこれをやっていた、という話あり。僕は LISP を使わなかったなあ。BASIC もインタプリタなんだけど、変数値の動的変更などは可能だったけどコードの修正をすると変数が失われるのが普通だったように思う。あれは変数スタック領域をコード領域のエンドから取ってたからか？確か整数・実数のメモリはエンドからで、再配置が必要になる可変長文字列変数がエンドからメモリを取り合うような仕組みになっていたせいだ。そうだ。そのはずだ。）

□ まとめ

ここまでで一段落した。つまり MVC 的なひながたクラス群を作って、最低限のメソッドをガワだけ実装してみた、という状態。
列挙するとこうか。

PaneModel
　クラスメソッド：new （ただしこれは New Class... メニューが自動的に作った）
　インスタンス変数：label, picture
　インスタンスメソッド：initialize （ただしこれは New Class... メニューがイニシャライザとして自動的に作った）
　　label, label:, picture, picture: つまりインスタンス変数のゲッターとセッター一揃い（これも New Class... が作った）
PaneView
　インスタンスメソッド：defaultControllerClass （単に自分のコントローラは PaneController である、と返すだけのもの）
　　displayOn: （画面（再）描画を行う）
PaneController
　インスタンスメソッド：redButtonPressedEvent:, yellowButtonPressedEvent: （クリック時に呼ばれるメソッド）
PaneExample
　クラスメソッド：example0, example1

これで一通りMVC 的にウィンドウを出すためのガワができたわけで、あとはこのガワの中身を実装する（つまりアプリ本来の仕事を実装する）ことになる。次回からはそうなるのであろう。。。

□ ウィンドウの中に文字を描いてみる

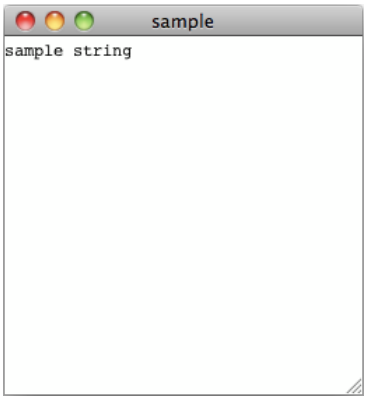
一段落した残り時間で、何かこれでやってみよう（少し中身を実装してみよう）という事になった。
何か無いかと聞かれて「ウィンドウの中にイメージを一つ表示する」という案が出たが、そこまで飛躍すると自分がついていけなくなるので飛躍を小さくしようと思って「文字を描画する」ことにしてもらった。

（「ウィンドウの中に何か描く」のがどのような構造になるか見たかったのだが、その前の「描くものの準備」の規模が大きくなると本来見たかった構造を見る余裕が無くなる、と思った次第。が、よく考えてみたらイメージひとつファイルから取ってきて描くだけなら、Smalltalk だと非常に隠蔽度が高く一行くらいでできたかもしれない。）

PaneView の displayOn: の中身を修正。（赤文字部分を追加）

```
displayOn: graphicsContext
    "自分自身を表示する。"
    | aComposedText |
    Transcript
        cr;
        show: thisContext printString;
        space;
        show: Time now printString.
    aComposedText := 'sample string' asComposedText.
    aComposedText displayOn: graphicsContext
```

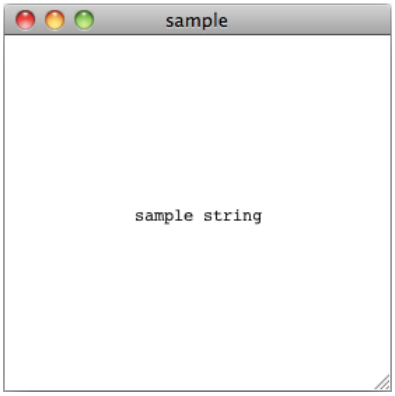
これで以下のようになる。もちろん Do it （アプリ再起動）なんかせずウィンドウサイズをいじったらちゃんと出てくる。



ComposedText てのは恐らく文字列に対してフォントなどの（文字のグラフィクス的な側面に関する）補助的な情報を持たせたものだろうなあ。
まあこれくらいの前処理で画面上に出るってのはステキ。
あと View に対する displayOn: が、その中身（パーツ）であるはずの文字（ComposedText）に対して同じ名前で利かせられる（多態性が利いている）というのは良いなあ。
つまりこういう構造にしたいんだなと言うことが判った。
また、そのために graphicsContext が渡されてきていて、またパーツの描画のためにパーツにこれを引き渡している。

次、表示される場所を変更する。画面ど真ん中でどうか。

```
displayOn: graphicsContext
    "自分自身を表示する。"
    | aComposedText |
    Transcript
        cr;
        show: thisContext printString;
        space;
        show: Time now printString.
    aComposedText := 'sample string' asComposedText.
    aComposedText displayOn: graphicsContext
        at: self bounds center - aComposedText bounds center
```

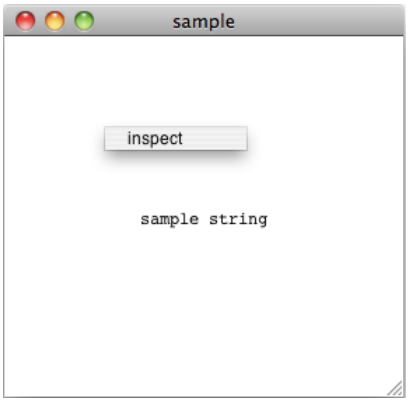


いしししし。

次、左ボタンを押した時の挙動をいじる。メニューを出してみるか。displayOn: ではなく今度は redButtonPressedEvent: を変更する。

```
redButtonPressedEvent: event
    "マウス左ボタンの処理について出す。"
    | aMenu |
    Transcript
        cr;
        show: thisContext printString;
        space;
        show: Time now printString.
    aMenu := Menu labelArray: #('inspect') values: #(#inspect).
    aMenu startUp.
    ^nil
```

実行（というかクリック）するとこんな感じ。



出る出る。

□ Save Image...

動作途中でも Save Image 可能。メモリダンプなので。
再度実行するとそのまま実行途中から再現される。

が、イメージ保存をし続けるとそのうちにイメージ（メモリ内容）が汚れていく可能性がある。綺麗に作業すれば問題ないけど。
てか Smalltalk は永続オブジェクトなので、やはり自分の環境でしか動作しないように準備してしまう可能性があるだろうなあ。
クリーンな環境であることが保証された状態で開発し続けた方が安全だ、という気分は良く分かる。