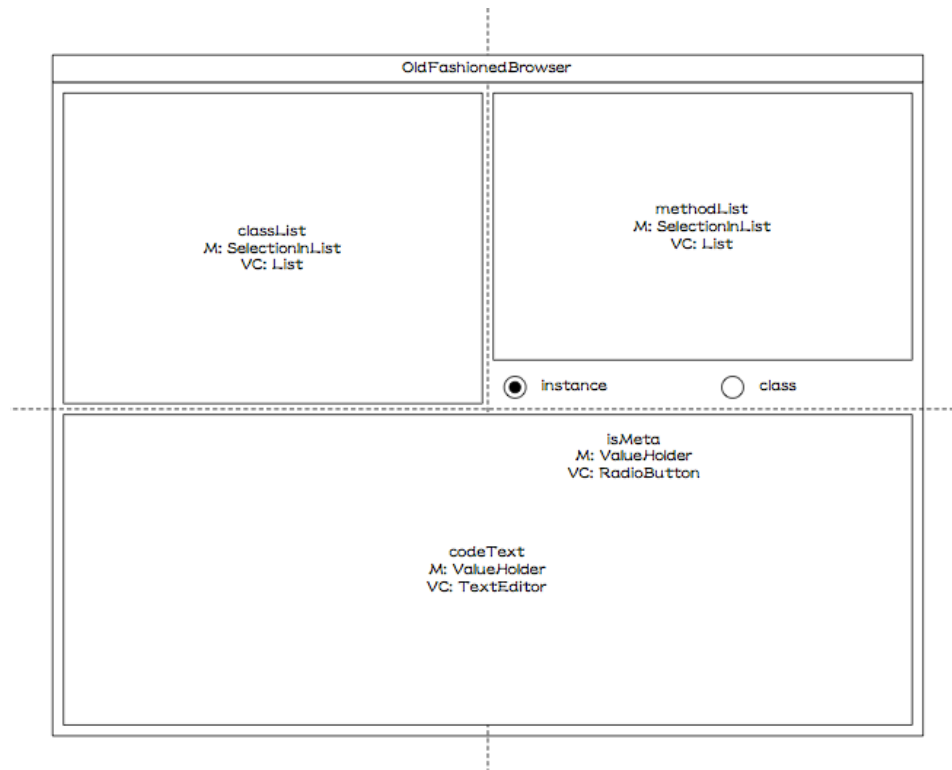


□ 2010.6.2

今回から新しい題材で。

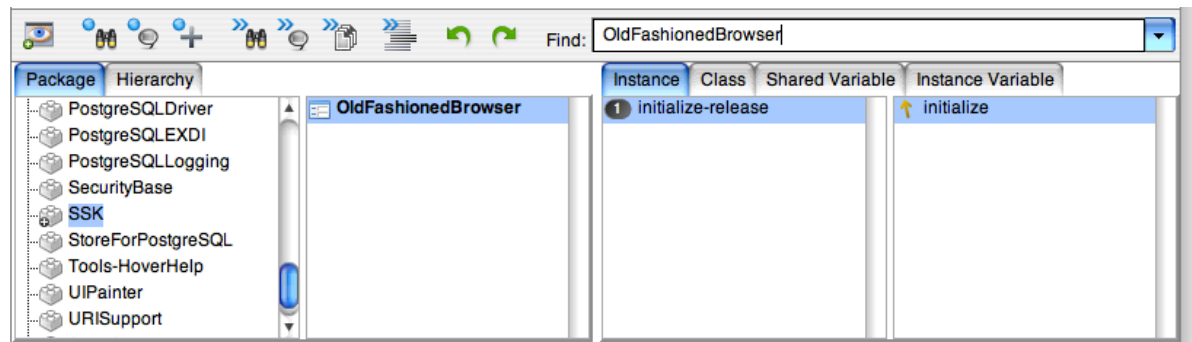
VisualWorks 77 nc のディレクトリにある SSK0.st を読み込ませて使う。

ファイルの読み込みは FileBrowser を操作して .st ファイルを選択し、コンテキストメニューから filein を選べばよい。



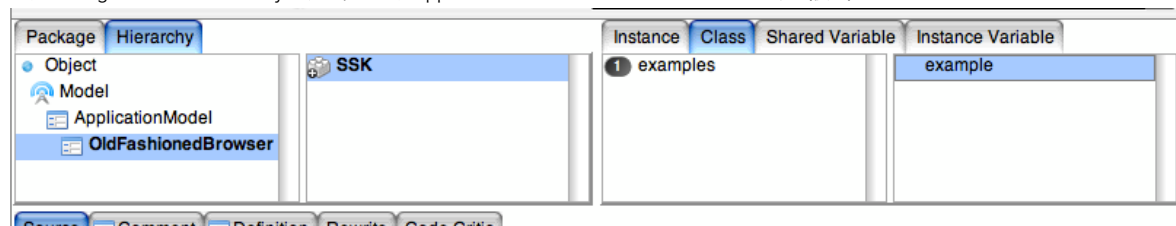
filein すると SSK というパッケージができる。

(なお filein したものがこれだよ、というようなメッセージなどは出ないらしい。.st ファイルの中身を見てなんとか勘を付けるしかないらしい。今回僕は以下のようにそれっぽい名前で見つけて、それが SSK パッケージである、ということを知った。)



□ 初期状態

で、Package ではなく Hierarchy の方を見ると、ApplicationModel の subclass になっている。(後述)



で、まず Instance 側、initialize メソッドを見る。



で見るとコードは以下の通り。

initialize

```
super initialize. <<< super initialize を投じているので、上述の ApplicationModel に initialize を投げる。
classTable := Dictionary new: 1000.
instanceMethodTable := Dictionary new: 1000.
classMethodTable := Dictionary new: 1000.
SystemUtils allClassesDo: << 全部のクラスのぶんだけ回る
    [:aClass |
        classTable at: aClass name put: aClass. <<<< 以下三つのディクショナリを作る。
        instanceMethodTable at: aClass name put: aClass selectors asSortedCollection.
        classMethodTable at: aClass name put: aClass class selectors asSortedCollection].
^self << で、自分自身（インスタンス）を return する。
```

ディクショナリは (key, value) でデータを引けるもの。

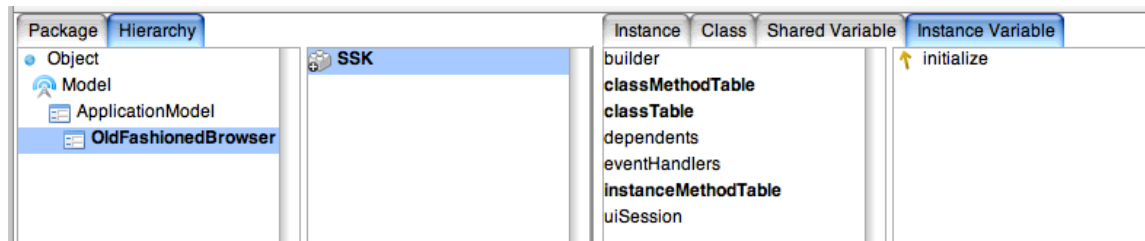
classTable は key=クラス名、value=クラスそのもの

instanceMethodTable は key=クラス名、value=セクタ群

classMethodTable は key=クラス名、value=セクタ群

のディクショナリになっている。

それぞれインスタンス変数として用意されている。



さて、このまま class method の中の example を見る。

example

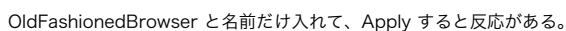
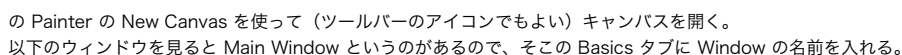
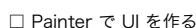
```
"OldFashionedBrowser example."
```

```
| aBrowser |
```

```
aBrowser := OldFashionedBrowser new.
```

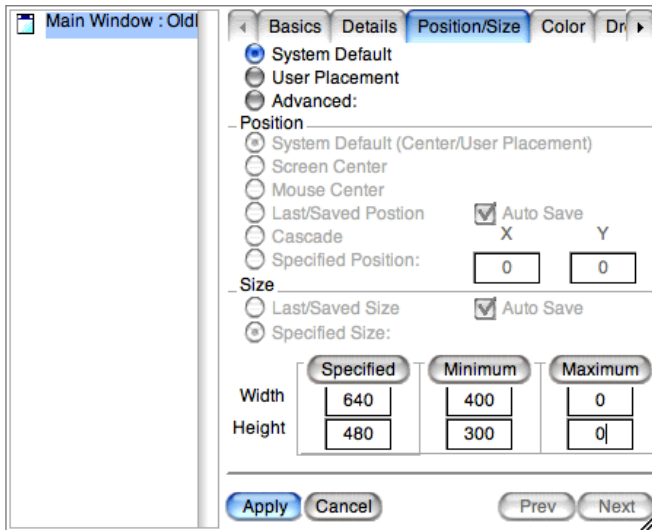
```
^aBrowser
```

この OldFashionedBrowser example. を inspect it すると、実際に三つのディクショナリに中身が入って出来あがった事がわかる。こんな感じ。

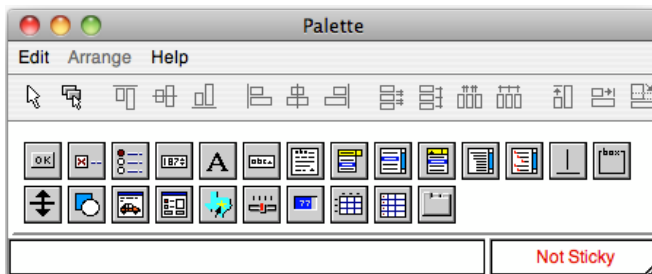


次、Position/Size タブで画面サイズ指定。

Specified はイニシャルのサイズ。Minimum は最低。Maximum が 0,0 なのは画面サイズ一杯まで、という指定。

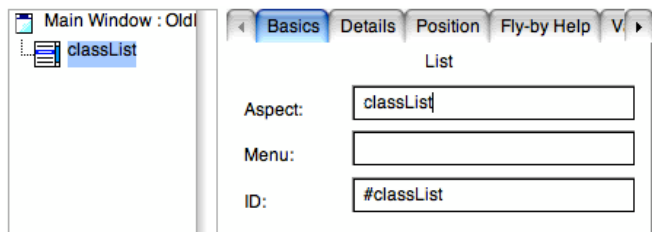


さて下のパレットからアイテムを選ぶ。



今回はまず List  を選んで、配置。

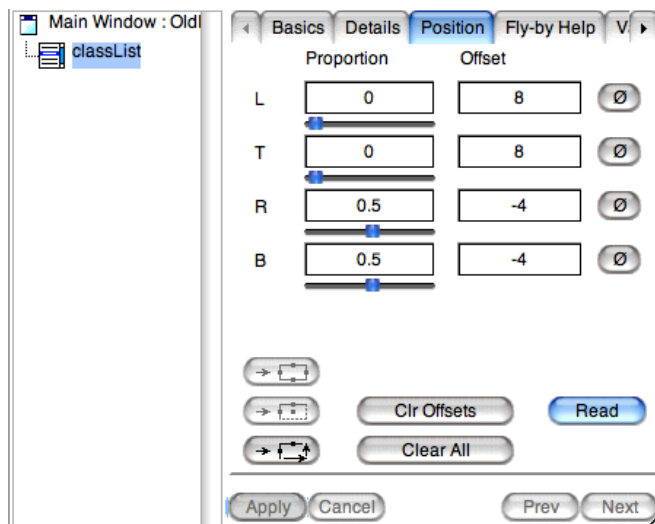
Aspect にメソッド（セレクト）名、ID は適当でいいんだろけど作法としてはメソッド名に # つけるとかするらしい。



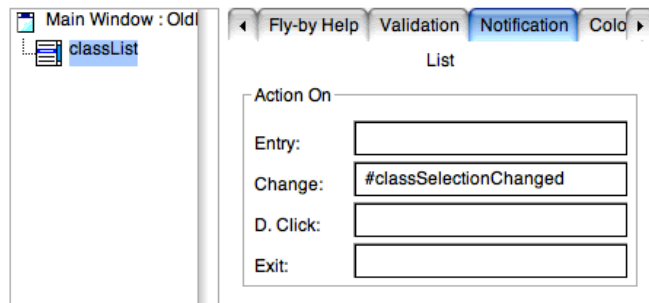
（と思ったら、あとで見ると Aspect の classList にも #classList と # がついていた。ふうむ。この classList という指定がどこで何になるのかは後述。）

次、Position で List のサイズを決める。これで Left Right Top Bottom についてプロポーションをどこにするか決めて、そこからのオフセットを指定する。これいいかも。

この List は左上の 1/4 に出すので、それで指定。



次、Notification タブ（スライドさせる必要あり）に以下のようにメソッド名（セレクト）を指定。これってつまり List について Change が起きたらこのセレクトを呼ぶというもの。

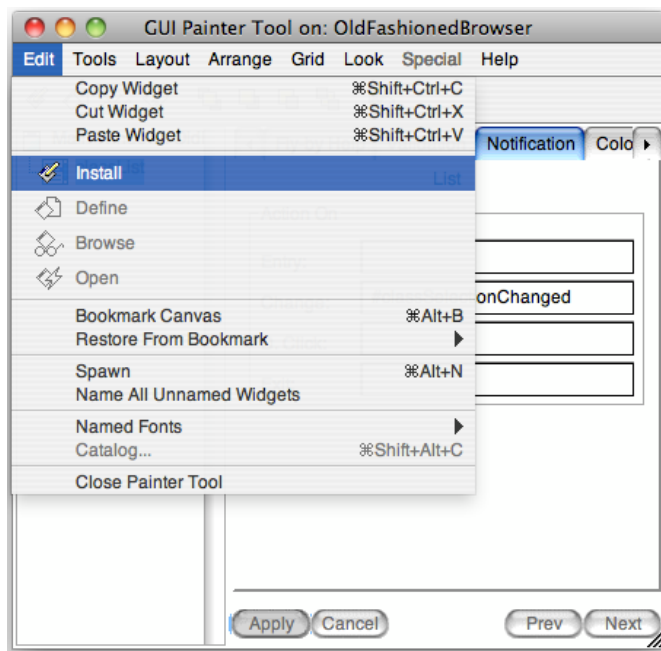


この Notification は、イベントが UI に対して発生した際に Application に送り出されるメッセージを設定する。
つまり List に対して change が出ると、対象となる Application（今回作ってるアプリそのもの、の、インスタンス）に向かって classSelectionChanged メッセージが発行される。
つまり先の ListView の Basics のところで Aspect に classList（実際には # がついて #classList）と入れたけれど、これはこの ListView のインスタンスが生まれた時にアプリに向かって投げるメッセージ名に相当する。
（内容的には Aspect なので「様相を示せ」かもしれないけど、ちょっと曖昧だ。コード（後述）を見ると、対応する Model を返せ、に読める。）

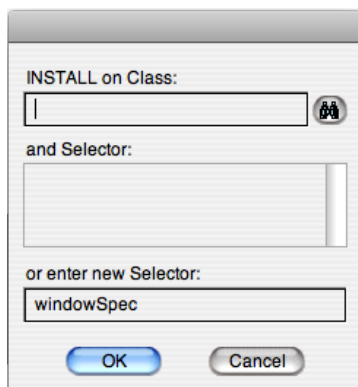
つまりこの ID #classList な ListView は、産み出されると classList を投げ、リストをつつくと classSelectionChanged を投げる。

なお Validation がほぼ同様の内容だけれど、Notification は「操作に由来するイベントが UI 上で発生し、システムに処理された後に出てくるイベント」を指す。Validation は「イベントが発生し、その処理がシステムでなされる前に出てくるイベント」を指す、らしい。


ここまでできたら Edit メニューの Install で書き出す。

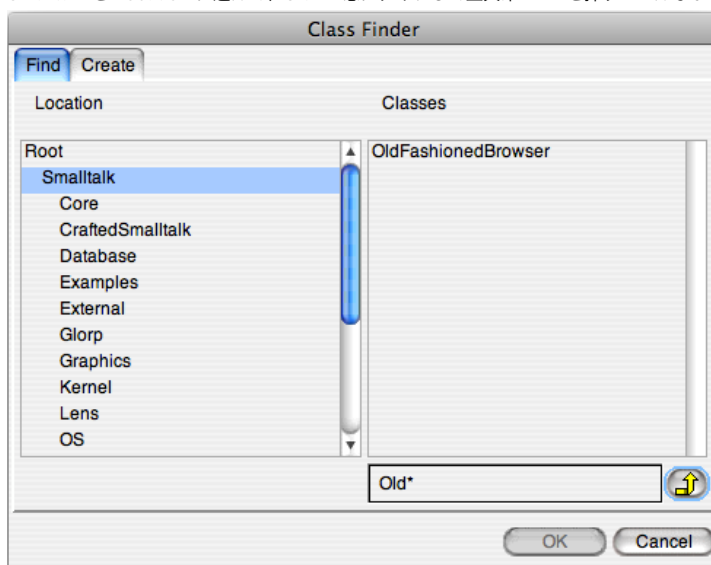


この UI を Install する先を指定してくれということでこんなのが出る。

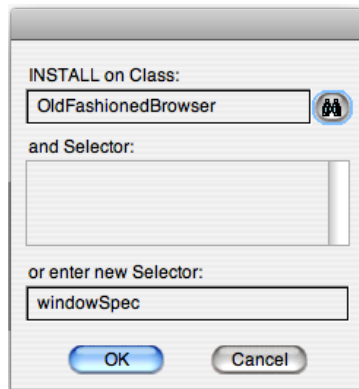


そこで双眼鏡をクリックして find 開始。

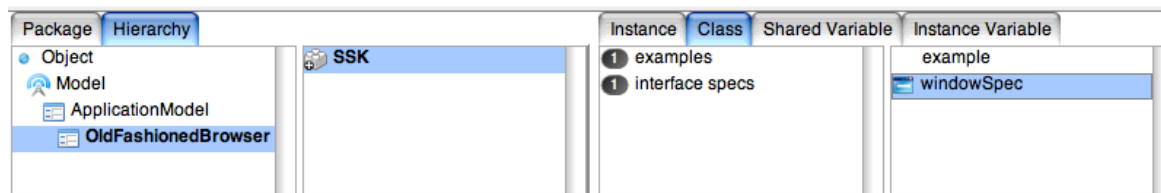
Smalltalk を Location に選んで、Old* と窓にタイプして上矢印  を押すとこんなふうに見つかる。



選択して OK ボタンをクリックすると、これが INSTALL 先指定の先ほどのダイアログに



これで Class 側に windowSpec というメソッドが出来る。



なおこの windowSpec のソースを見るとこんな感じ。
定義した情報がそのまま入ってる。

windowSpec

"Tools.UIPainter new openOnClass: self andSelector: #windowSpec"

```
<resource: #canvas> <<<<< ★★ これ、コンパイラ指示子だなあ
^#({UI.FullSpec}
  #window:
    #({UI.WindowSpec}
      #label: 'OldFashionedBrowser'
      #min: #({Core.Point} 400 300 )
      #max: #({Core.Point} 0 0 )
      #bounds: #({Graphics.Rectangle} 542 233 1182 713 ) )
    #component:
      #({UI.SpecCollection}
        #collection: #{
          #({UI.SequenceViewSpec}
            #layout: #({Graphics.LayoutFrame} 8 0 8 0 -4 0.5 -4 0.5 )
            #name: #classList <<<<<<< ★★ 私の名前は classList である
            #model: #classList <<<<<<< ★★ モデルは classList である
            #callbacksSpec:
              #({UI.UIEventCallbackSubSpec}
                #valueChangeSelector: #classSelectionChanged ) ★★ 変更には classSelectionChanged を発行
            #useModifierKeys: true
            #selectionType: #highlight ) ) ) )
```

この windowSpec は普通にはコンパイルされず、この定義に従う View + Controller を作ってくれる、という上位にあるのか。
ともあれ使えるように用意してくれる、と。

で、model は classList だよ、となっており、これは Aspect で入力したものそのものだ。

つまりこの UI (=View+Controller) が対象とすべき Model は classList である、ということになる。

実際には self に対して classList メッセージを投げて、それで得られるオブジェクトが Model であるはずだ、という構造に見える。
(実際にこの classList メッセージはこの ListView が扱うべき (リストの中に収める) モデルを返す。)

□ モデル側 (自分のアプリ側) の準備

こうして作った UI と、自分のアプリとの接続はメッセージ名で行う。

つまり ID #classList な ListView は、産み出されると classList を投げ、リストをつつくと classSelectionChanged を投げる。

そこで initialize に以下のように classList オブジェクトに SelctionList new を突っ込むようにする。

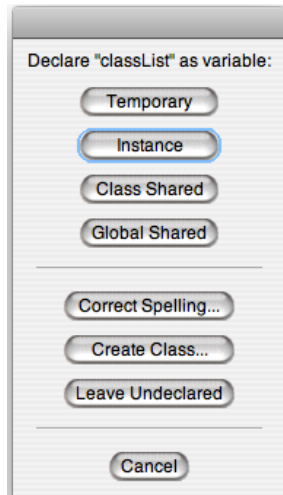
initialize

super initialize.

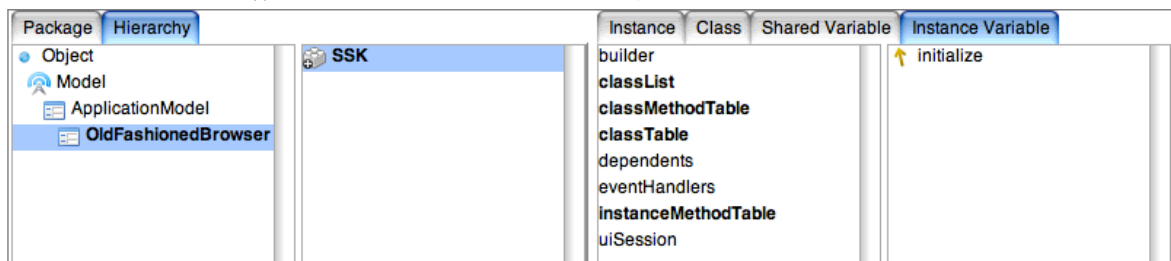
classTable := Dictionary new: 1000.

```
instanceMethodTable := Dictionary new: 1000.
classMethodTable := Dictionary new: 1000.
SystemUtils allClassesDo:
    [:aClass |
        classTable at: aClass name put: aClass.
        instanceMethodTable at: aClass name put: aClass selectors asSortedCollection.
        classMethodTable at: aClass name
            put: aClass class selectors asSortedCollection].
classList := SelectionInList new. <<<<<<<< これを追加
^self
```

これで Accept してやると以下のように classList はない、と言う。

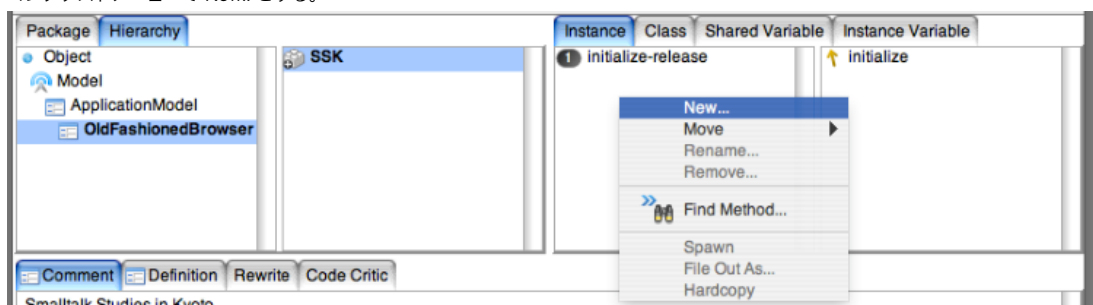


そこで Instance variable だと言ってやると、definition にできている、らしい。



おお、classList がインスタンス変数として（さっきの三つに）追加されている。

Instance のカテゴリ（プロトコル）として aspects というのを作るう。
コンテキストメニューで New.. とする。



ダイアログを出して名前を聞いてくるので aspects と指定して、新しく作る。
そこで classList, classSelectionChanged の二つのセレクタを追加する。

まず classList メッセージは（Window が開かれ、続いて）ListView が開かれた時に送られてくる。
コードは以下の通り。

```
classList
    classList list isEmpty ifFalse: [^classList]. << 最初に空か否かをチェックする
    classList list: classTable keys asSortedCollection. << 空だったら内容をソートしてインスタンス変数に保持しておく
    ^classList
```


ところでインスタンス変数 classList は initialize の際に

```
classList := SelectionInList new.
```

として設定されているはず。つまり、

- 1.) classList オブジェクトの list 部分については（他に何かがあるか知らないが）まず new されて Empty であるが
- 2.) これについて classTable テーブルに格納されたクラスのディクショナリのうち、そのキーだけを sortedCollection として取りだしたものを list 部分にクラス名の文字列のリストを設定する仕事をする。

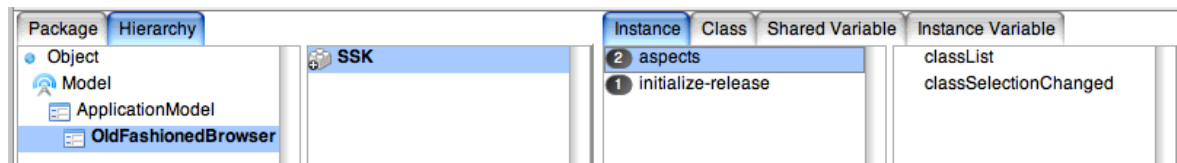
次、classSelectionChanged メッセージは、何か ListView に変化があったときに送られてくる。

コードは以下の通り。

```
classSelectionChanged
  Transcript
    cr;
    nextPutAll: thisContext printString
```

単にクリックされた場合はちょっとトランスクリプトに出す、と。(thisContext の意味がようわからんが今は無視)

これでインスタンスメソッドが二つ用意された。



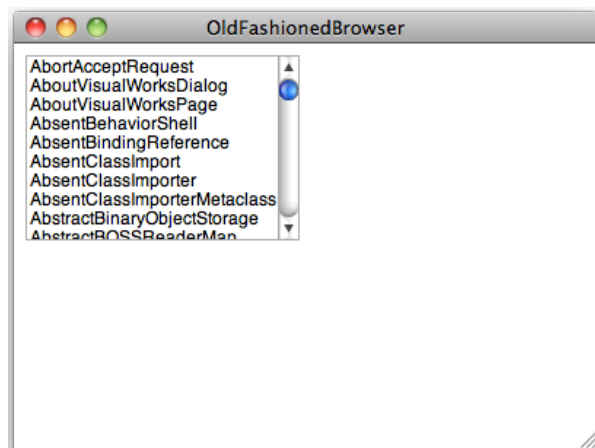
□ ブラウザを開けてみる

で、example に aBrowser open を追加する（ブラウザを開け、と言う）。

example

```
"OldFashionedBrowser example."
| aBrowser |
aBrowser := OldFashionedBrowser new.
aBrowser open. <<<< これを追加する
^aBrowser
```

で、期待通りブラウザが開いた。



これで問題無く動くようになったが、例えばクリックするとクラス名だけでなく情報ごと出すようにするにはこんな風に。

```
classSelectionChanged
  Transcript
    cr;
    nextPutAll: thisContext printString;
    crtab;
    nextPutAll: classList selection printString;
    flush
```

□ UI (View+Controller) と Model との接続

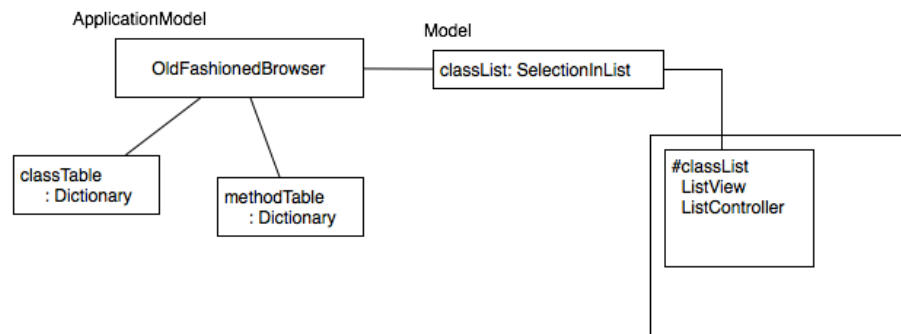
ところで initialize に突っ込んだ一行に注目。

```
classList := SelectionInList new.
```

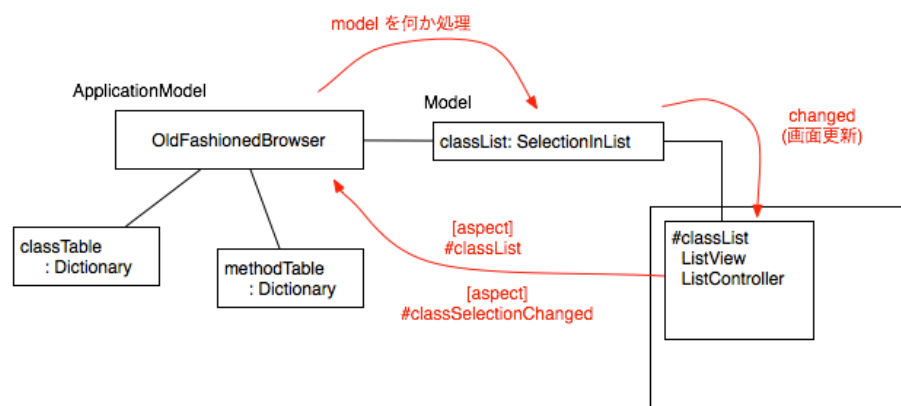
この SelectionInList は painter (UI builder) が置いた ListView (VC に相当) に対応する Model として予め対応すべく定義されたもの。

ListView にはその対応する Model として SelectionInList を使いなさい、ということになっている。
(この知識はライブラリのマニュアルとして提供される。(あるいはソースを見る))

で、その SelectionInList を classList というオブジェクトとして用意しておけば、あとで classList に対して View+Controller が相手してくれるから、ということだが、、、
以下各種オブジェクトの配置。



で、その builder が作った UI (ID : #classList の object) がどこで Model を指定しているかということ、先から挙げている Basics の Aspect 項目に入れた (そのせいで windowSpecs では model: として現れていた) classList という名前。
例えば Window を開き、問題の ListView が開かれた時に、ListView はアプリに対して「model を教える」と言う。(Application に対して Model を要求するメッセージ (名前は classList と指定した) として発行される。以下のような感じ。
(赤線は下記の説明の流れを示すだけに過ぎず、厳密にメッセージやデータの流れを示すものではない)



まず一番下の #classList メッセージがアプリに対して発行される。
受け手である classList メソッドには以下のように書かれており、、、
1: classList
2: classList list isEmpty ifFalse: [^classList].
3: classList list: classTable keys asSortedCollection.
4: ^classList

行番号に沿って解説すると、

- 2: インスタンス変数 classList が初期化済みならそれ (中身のある SelectionInList のインスタンス) をそのまま返す
- 3: 初期化がまだなら他のインスタンス変数 classTable (クラス一覧辞書) からキーだけ取りだし、キーだけソートして、それを SelectionInList の list 部分に格納して
- 4: classList つまり中身をちゃんと詰めた SelectionInList そのものを返す

つまり「Model を我に示せ」と要求した classList メソッドに応えた事になる。
また、このとき Model に対してなんらか操作をしている。(まあ別にせんでもええけど)
そして View + Controller に対して、データが更新されたから何かせえよ (再描画せえよ) と言っている。
(この再描画については次の methodList の更新処理の方がわかりやすいか)

うーん、そこらじゅうに classList という名前が埋まっているので、どれがどれと絡んでいるのか、単にかぶった名前がついただけなのか分からないなあ。。。Model の classList (インスタンス変数) は上の説明では ListView が直接知らなくても良い (アプリに対するメッセージパッシングと return の結果でもってデータの入出力をすればいい) のだけれど、果たしてそうになっているかどうかは分からないなあ。。。

もう一つ。SelectionInList を使いなさい、という事は良いとして、その知識をマニュアルから得るのは良いとして、SelectionInList には list: という要素 (?) があり、そこに ListView に表示される文字列のリストを入れておくと良い、という知識は誰がどこから仕入れるのか?

聞くと、どうもこれはマニュアルに載っていないらしく、ソースコードを見て勘を付けるような状況でやるらしい。
まあしかしそれでもとにかくそのスジが通れば、プログラマとしては何をどこに用意すれば接続が通るのかははっきりする。

とりえず経路は見えた。

なお今回 ApplicationModel の subclass となっていたが、これはつまりこうした Model ばかり集めたものを作る、それが実は Application なのだ、というアイデアから来ている。
これはちょっと良い視点と思う。

□ もう一つ ListView オブジェクトを置いたら？

もう一つ ListView を置く。

つまり

- ・ UI builder で ListView を追加
- ・ 名前を methodList, Action の Aspect に #methodList, Changing に #methodSlectionChanged を登録
- ・ Install で書き登録

して、

- ・ インスタンスメソッドとして methodList, methodSelectionChanged を用意
- をする。

initialize

....

```
classList := SelectionInList new.  
methodList := SelectionInList new. <<< これを追加  
^self
```

で、builder で同じようにもう一つの List を設置。Window の右上に。

再び methodList, methodSelectionChanged メソッドを追加する。

methodList

```
| aSymbol |  
(aSymbol := classList selection)  
ifNil: [methodList list: SortedCollection new] << 選択されていなければ空で用意  
ifNotNil: [methodList list: (instanceMethodTable at: aSymbol)]. << 選択されていたら instanceMethodTable から引く  
^methodList
```

--- えーと

instanceMethodTable は initialize でこんなして作りましたね

```
instanceMethodTable := Dictionary new: 1000.
```

.....

```
instanceMethodTable at: aClass name put: aClass selectors asSortedCollection.
```

たぶんどっさりできてるのでしょうね。

----- おわり

methodSelectionChanged

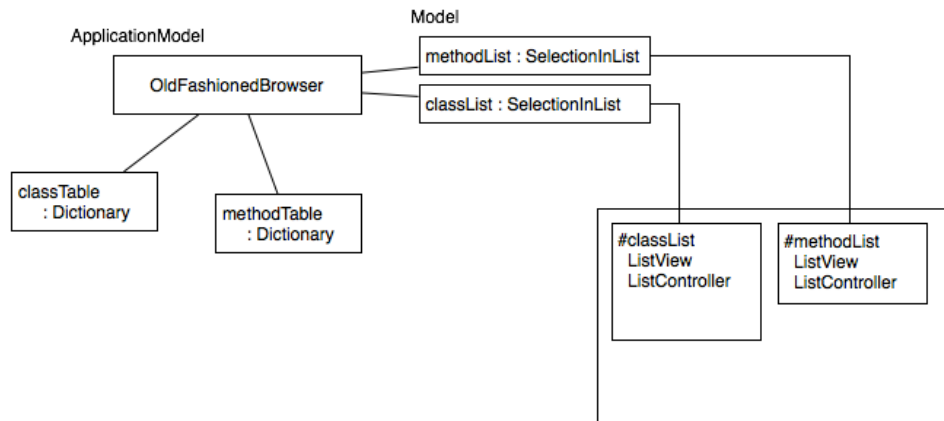
```
Transcript  
  cr;  
  nextPutAll: thisContext printString;  
  crtab;  
  nextPutAll: methodList selection printString;  
  flush
```

さて、classSelectionChanged の method を Transcript 出力ではなく、この methodList を投げるようにする。

classSelectionChanged

```
self methodList.
```

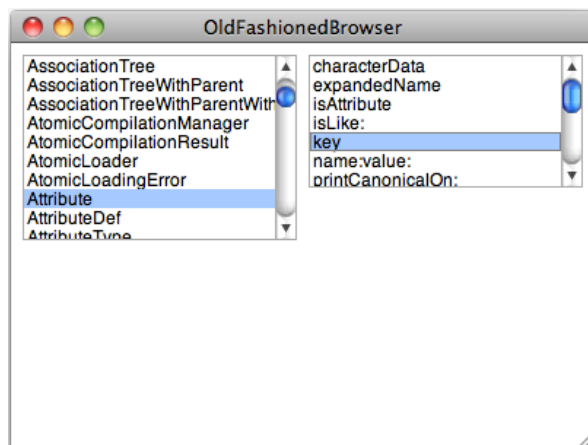
これで以下ようになったはず。



クラスが変わるとメソッドリストが変わるところが面白いので以下に流れを。

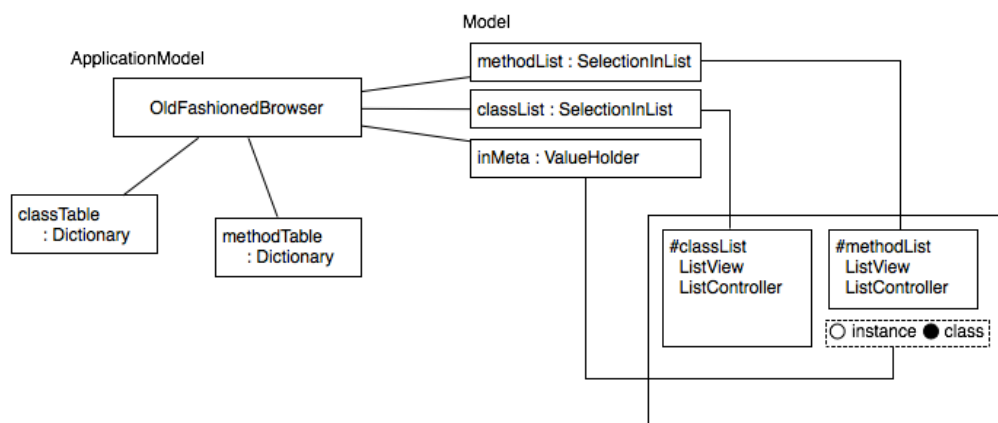
1. classList の ListView をクリックすると、classSelectionChanged メッセージが発行され、
2. アプリはこれを受けると self に methodList と投げて、
3. methodList に対応するはずの Model を再設定して
4. そのあと再描画(changed)が投げられているはず

これで再び OldFashionedBrowser example. とすると、以下のように表示されてちゃんと動作している。



□ 今度はラジオボタン

ラジオボタンを追加すると、以下のような構造になるはず。



さてまたしてもやることは同じ。

1. 用意する radio button の Model に相当する Instance 変数 (isMeta) を用意して、
2. それを取得するメッセージ isMeta を用意する。

まず 1. は初期化が必要なのでそこで作る。

```

initialize
    super initialize.
.....
classList := SelectionInList new.
methodList := SelectionInList new.
isMeta := ValueHolder with: false. <<< これを追加。今度の radio button のモデルは ValueHolder なのだ。
^self

```

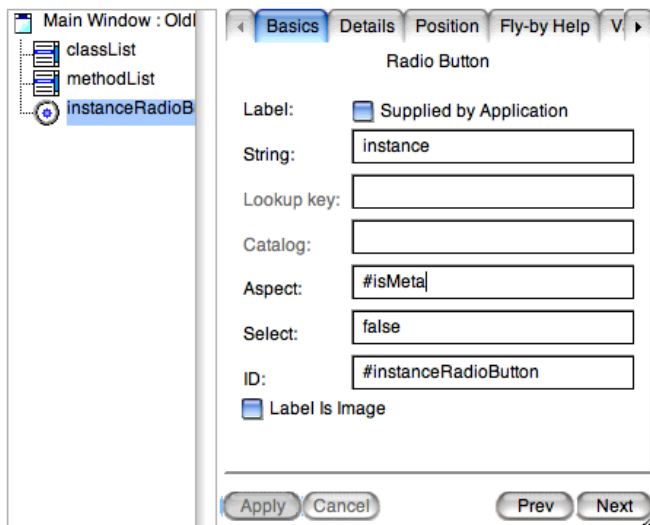
2. は作成する Radio ボタンが isMeta メソッドを呼んでくれるようにする。
 そのせいで isMeta メソッドは作る必要があるが、なんにもせず ^isMeta するだけで良い。

```

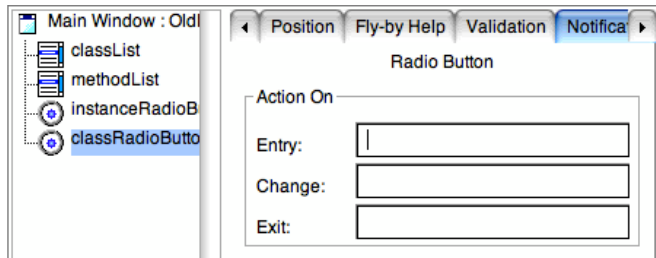
isMeta
    ^isMeta

```

UI builder で Radio Button を設置するようす。

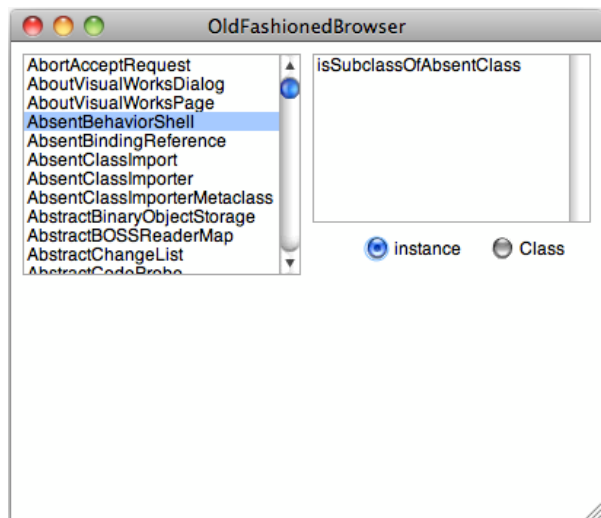


もう一つ、class ボタン側も上とほぼ同じで作る。違うのはラベル文字列と、Select が true であること、か。
 Radio Button にも Notification で Change があるが、今回はなんにもしなくても動いている。



Select は「自分は Aspect によって返ってきた Model (の with: で設定したもの) が「コレ」だったら点灯する」ように振る舞う。
 if 文の条件判定の片側だと思えば良い。
 そのため、これを true/false の二値にせず label で 1, 2, 3 or a, b, c などとしてもよいとのこと。
 それで二つより多いラジオボタンの選択処理などができるという。(未確認、というかソースでは Boolean, Fraction, String 対応ばい)

とりあえずこれだけでちゃんと on/off 相互反転が利いている。



(ということは実は今回 Aspect のメッセージを投げている時に、実質 Select の値も with: 添えて投げつけるようなことをしている？
誰かが isMata インスタンス変数の値を変えてやらねばならんからねえ。
普通に考えると isMetaChanged メッセージに isMeta の反転処置 (値の設定) を置くよねえ。
isMeta selection みたいにして、ValueHolder に何かさせるよなあ。ValueHolder には setValue: てるがあるぞ。単に代入するだけだが。)

さて instance と class の切り換えをせにゃならん。宿題ね。