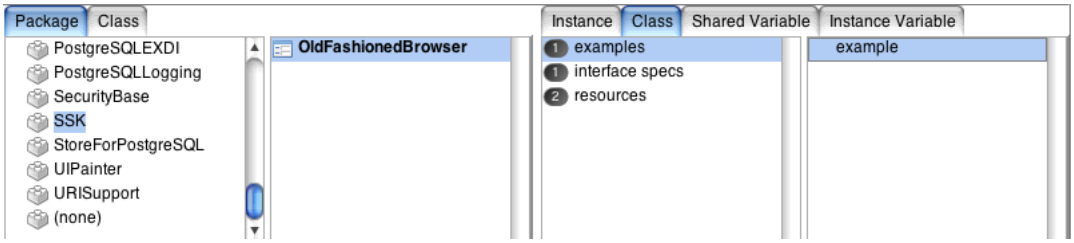
□ 2011.2.2

SSK10.st を File in （ファイルブラウザで開いて、コンテクストメニューから操作。いつも忘れる。）



これが入っているはず。

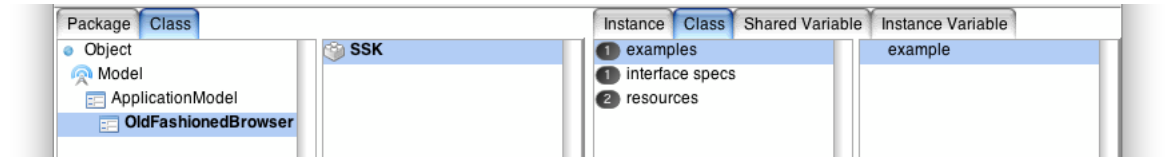
ところで最近は一時変数を書かないなあ、と。example には以下のように書かれている。

```
^(OldFashionedBrowser new)
    open;
    yourself
```

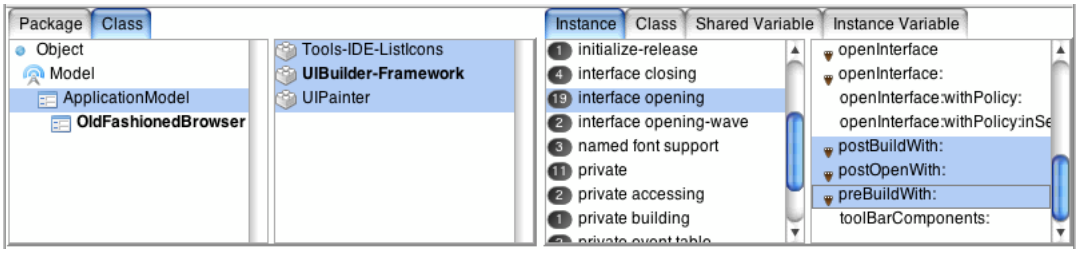
＃テンポラリを使わない書き方を今回は何度もやった。

今回はキー入力があったら最初は inactive になっていた accept メニューが enable になるような、そういうことをやろうと。
（キー入力をキャッチするような処理、と、メニューの中身（あるいは状態）を操作する、ということの二本立て。）

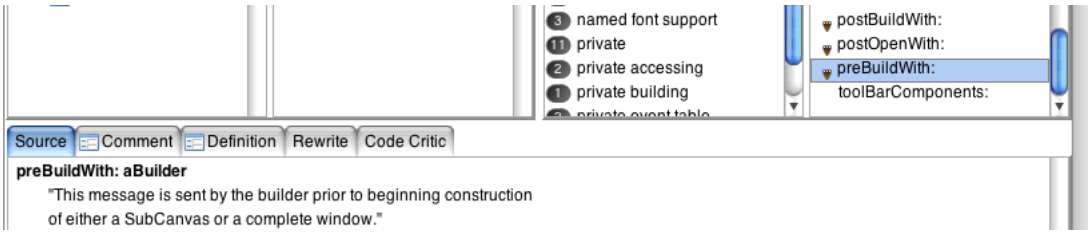
まず階層構造をチラ見。上位は ApplicationModel である。



一つ上の ApplicationModel のインスタンス側を見ると、この三つが見つかる。



ところがこのメソッド、どれも実装は空だ。



三つとも空なんだけど、つまりこれはフックルーチンじゃないの。
（と思ったら後で Hooking と紹介された。）

postBuildWith: aBuilder
"This message is sent by the builder when it has completed work on either a complete window or a SubCanvas."
このメッセージはビルダーから送られる。window か SubCanvas として ちゃんと働くようになったらね。（ビルドが終わったら、かな）

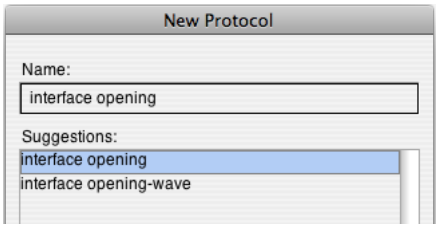
postOpenWith: aBuilder
"This message is sent by the builder after it has opened a completed window."
このメッセージはビルダーから送られる。まともな window として開いた後にね。

preBuildWith: aBuilder
"This message is sent by the builder prior to beginning construction of either a SubCanvas or a complete window."

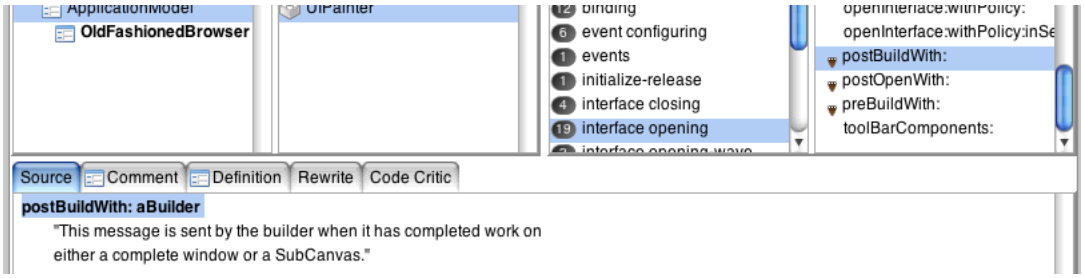
このメッセージはビルダーから送られる。まともなwindow か SubCanvas として
コンストラクトされはじめる、その前にね。

ここで出てくる「ビルダー」というのは、このウィンドウなどの構成要素をビルドしてくれるもの、のことらしい。
（もうちょっといい名前無いのか？と一瞬思うなあ、、、固有名詞でなく一般名詞なのか？）

実際これを作ってみる。まずは SSK のinstance 側でプロトコルとして interface opening を作る。
コンテクストメニューで new とやれば候補が挙がる。



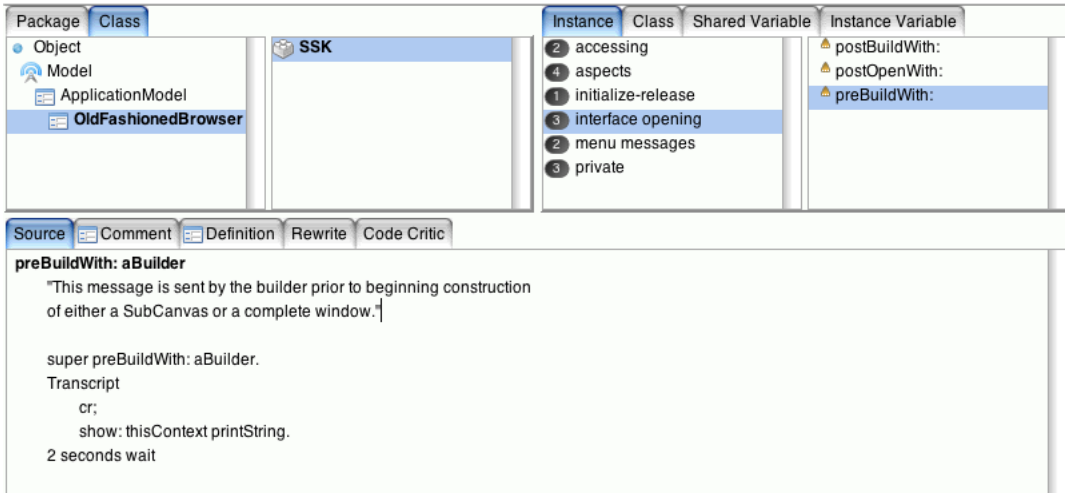
これに上記三つのインスタンス・メッセージのセレクタを作るが、中身をまず空で作る。



一つ上のApplicationModel から、メッセージごとに取ってきて単に SSK のインスタンスメッセージの欄に貼り付けて accept すればよい。

次にエイヤと中身にデバッグ出力を入れてフックが掛かっていることを確認する。

```
preBuildWith: aBuilder
	super preBuildWith: aBuilder.
	Transcript
		cr;
		show: thisContext printString.
	2 seconds wait
```



Transcript には以下のように出る。

```
|
OldFashionedBrowser>>preBuildWith:
```

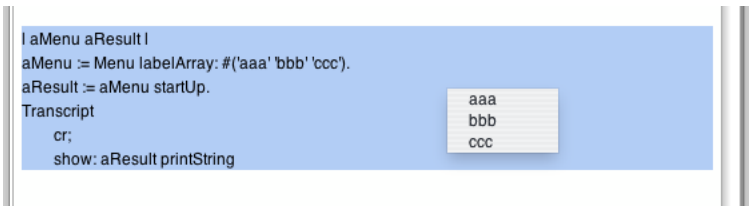
こんな感じで実行するとどのタイミングで出るかが観測できる。（特に 2 秒とか待たせておけば順序が明確になる）

次にワークスペースで実験。メニューが出てくる単純なケースを試す。

```
| aMenu aResult |
aMenu := Menu labelArray: #('aaa' 'bbb' 'ccc').
aResult := aMenu startUp.
Transcript
	cr;
```

show: aResult printString

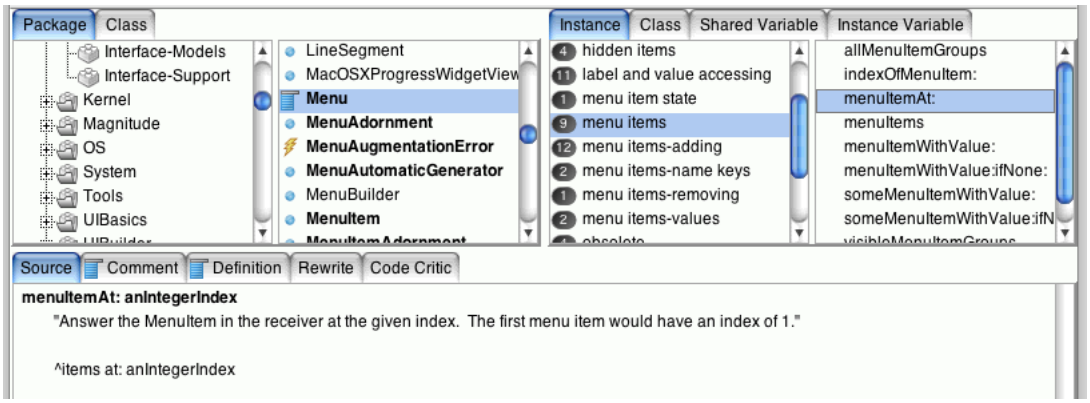
こんなメニューが出てくる。



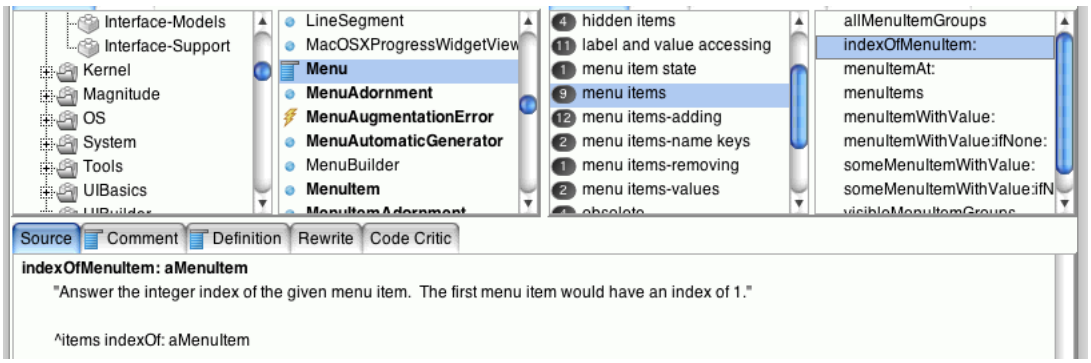
で、選択されたメニューのアイテムに対する処理（例えば今回はこれを disable したい）としてはこういうコードを想像するだろう。

aMenu := Menu labelArray: #('aaa' 'bbb' 'ccc').
anItem := aMenu zzzz: 'bbb'. << メニューのアイテムを（ラベル名を指定するなりして）取り出して、
anitem xxxx. << それに対して何かをする。（例えば disable にする、とかね）

さてブラウザで探索すると、

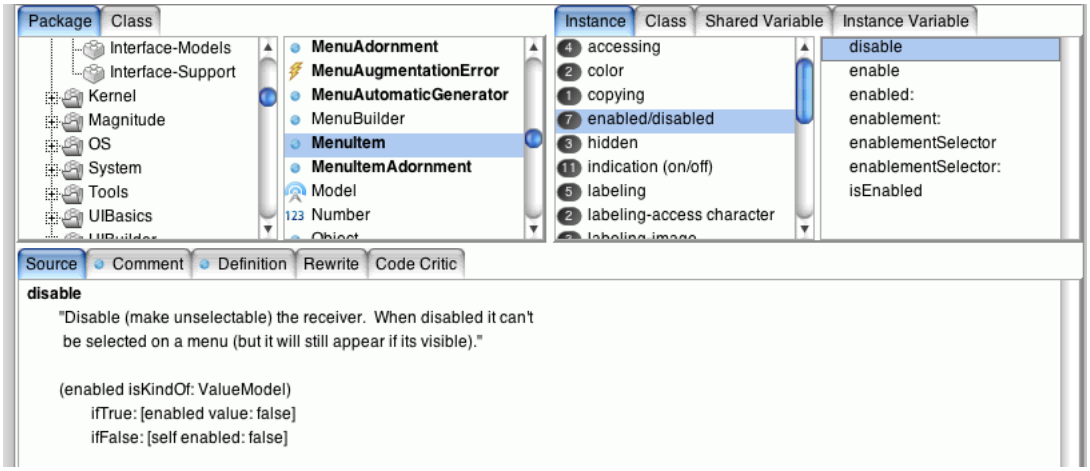


こう言うのがある。すぐ上にはその逆っぽいもの indexOfMenuItem: がある。



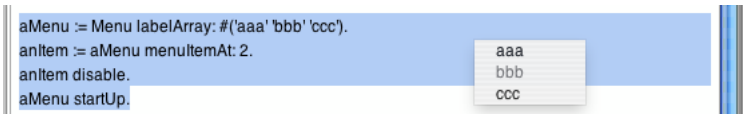
まあこれでよかろう。

では次にこのようにして特定されたアイテムを disable にしよう。
MenuItem てのにそのものずばり、がある。



これか。では先の「ありそうなコード」は具体的にはこうか。
aMenu := Menu labelArray: #('aaa' 'bbb' 'ccc').
anItem := aMenu menuItemAt: 2.
anItem disable.

Workspace で試す。ちゃんと disable された。

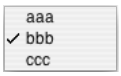


メニューラベルは多国語になる場合が多いので、コードとしては間接的に書く方が良い。value を別に与えてそれで処理しよう。

aMenu := Menu labelArray: #('aaa' 'bbb' 'ccc') values: #(#a #b #c).
anItem := aMenu menuItemWithValue: #b.
anItem disable.

でいいか。（後で Workspace で動作を確認した）

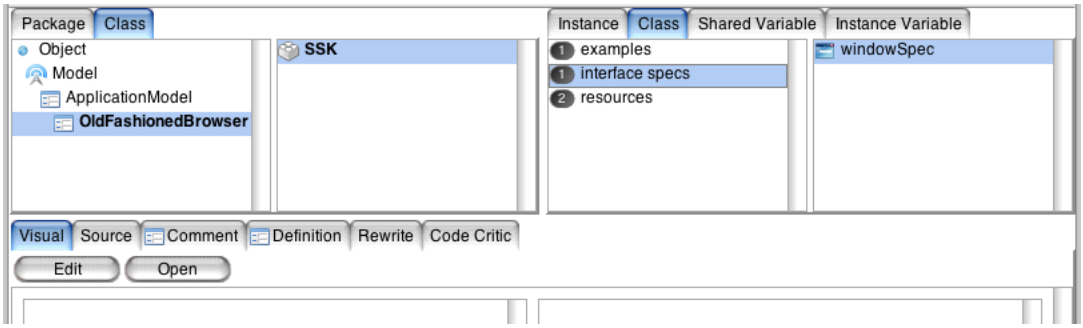
anItem hidden: true. ではメニューから消えてしまう
anItem beOn. チェックマークをつける



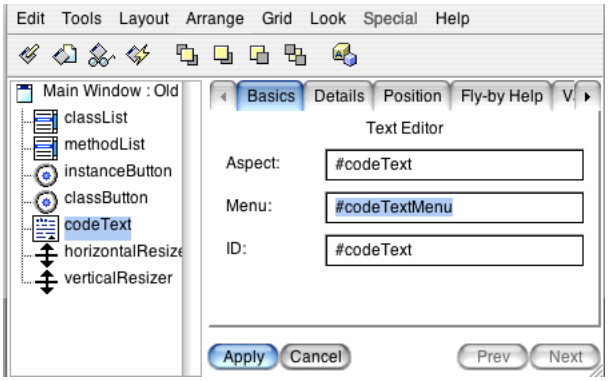
などもある。

今回はコードテキストを出すフィールド（下図の下半分にある CodeText 部分）にコンテキストメニューを追加する。

クラス側にある windowSpec を選び、



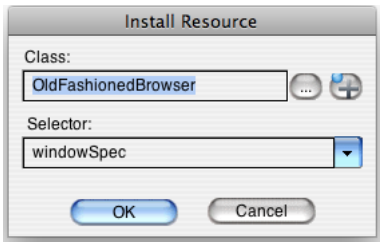
GUI ツールの Edit ボタンを押して修正。
下図のように codeText を選び、Basics タブを開いて、



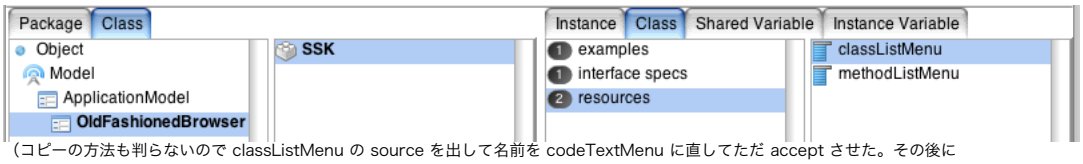
その Menu のところに #codeTextMenu と入れる。

これは「codeText フィールドで Menu の要求が出たら（コンテクストメニューを出そうとしたら）、codeTextMenu メソッドを呼んでくれ、という意味。
（当初は何も定義されていないのでデフォルトのメニューが出る）

で、Apply ボタンを押し、Edit メニューの Install を実行してインストールする。

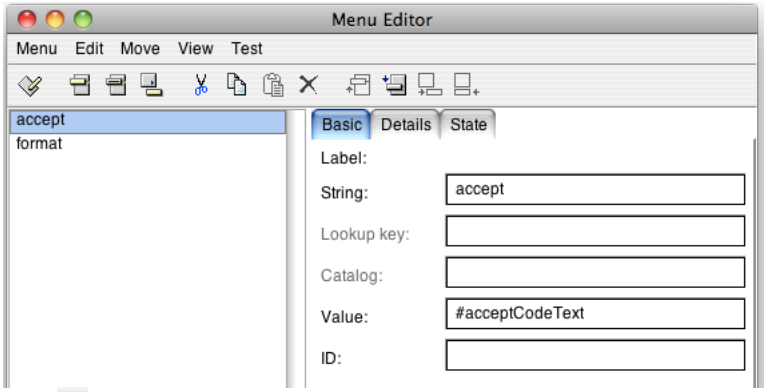


次に実際にメニューを追加する。呼び出し名は codeTextMenu だった。
が、ここでどうやったら新しいメニューを追加できるのか GUI 的操作方法が判らなかった。（ノートが間に合わず見逃した）
既存のメニューは以下にあるので、今回はエイヤとこれをコピーすることにする。

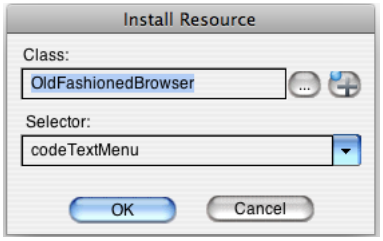


（コピーの方法も判らないので classListMenu の source を出して名前を codeTextMenu に直してただ accept させた。その後に
できた codeTextMenu に対して Visual タブを選んでエディタを表示させ、accept と format という項目をただ入れた。）

##このとき僕は呼び出し（callback）メッセージを acceptCodeText: だけにしており、青木サンプルでは acceptCodeText: from: と
##なっていた。方法を見落としている（ここの操作は追いついていなかった）ので、僕は from: ナシでやっている。（★from:抜け）



ここで  ボタンか、Menu の Install を実行する。以下のダイアログが出る。



これで実際にメニューができると codeText でメニュー操作をすると、作ったメニューが（常時）表示される。

さて実際にメニュー項目を選ぶと指定したメソッドが呼ばれるので、その対象をインスタンスメソッドの menu messages に登録しておこう。

acceptCodeText: aText from: aController
```
Transcript cr; show: thisContext printString
```

formatCodeText: aText from: aController
```
Transcript cr; show: thisContext printString
```

（すぐ上の ★from:抜け のところで書いたが、僕は from: 引数の設定方法について見落としているので、ここは from: なしで
書いておかないと実際に accept メニューを実行したときに（そんなメソッドは知らないと）エラーが出る。）

今回はこうやって作った accept メニューを、
1. 最初は disable 状態にしておいて、
2. codeText にキー入力があったら
3. enable 状態にする、という事をめざす。
この 2. の処理は先ほどのウィンドウのビルダーに対するフックルーチンを使って行う。
具体的にはキー入力に対するフック処理を仕掛けることになる、、、妙な入れ子だ、、、つまり
「ウィンドウのフックルーチンの中に、『キーイベントのフックルーチンを登録する処理』を登録する」
ことになる。

それには codeText に対するコントローラを得る必要がある。（キー入力はフィールドのコントローラがフックを用意している）
まずはビルダーのフックルーチンに対して、以下のようなことをして Wrapper というのを見てみる。

postBuildWith: aBuilder
```
| aWrapper |
super postBuildWith: aBuilder.
aWrapper := aBuilder componentAt: #codeText.
Transcript
    cr;
    show: aWrapper printString.
```

として、実行。componentAt: で codeText について限定している。Transcript には以下のように出る。

OldFashionedBrowser>>preBuildWith: << これはさっき試したデバッグ出力の残り
a SpecWrapper on: a BoundedWrapper on: a BorderDecorator << これが問題の aWrapper の内容

aWrapper は SpecWrapper だと思って良いか。これについてはインスペクタで中身を少し見させて貰えたがよく見ていられなかった。
（つまり WindowSpec からゴチャゴチャとビルドした結果、あれこれの処理層でくるまれたものではないか）

そもそもこのフックルーチンには aBuilder が渡ってくるが、これ、ApplicationModel のインスタンス変数として存在していたりする。
（つまりこのアプリでもそれは見えている（あるいは自分で持っている）はず。）
それをわざわざ引数で渡している。おもろい。ビルダーごといじってくれということか。

一時変数が無いように、以下のように書き換えてみる。（あまり本質的ではないけれど、、、）

postBuildWith: aBuilder
```
super postBuildWith: aBuilder.
```

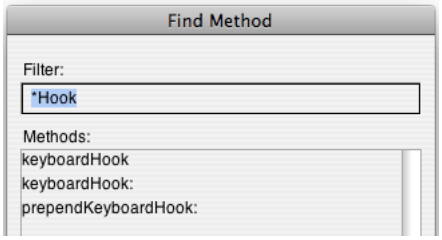
```
(aBuilder componentAt: #codeText)
    ifNotNil:
        [:aWrapper
            | Transcript cr;
            show: aWrapper printString]
```

さて
aWrapper widget printString
とやると、MVC の V が取り出せる。Transcript への出力としては
a TextEditorView
と出る。では、
aWrapper widget controller printString
とやれば当然、
a TextEditorController
となる。これで念願の codeText のコントローラ（TextEditorController）が手に入った。

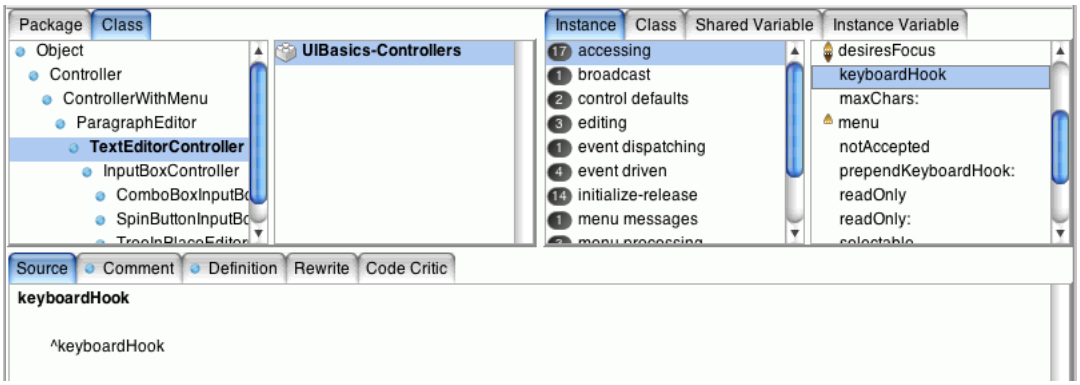
postBuildWith: aBuilder

```
super postBuildWith: aBuilder.
(aBuilder componentAt: #codeText)
    ifNotNil:
        [:aWrapper |
        Transcript
            cr;
            show: aWrapper widget controller printString]
```

TextEditorController でオブジェクト検索する（単にブラウザ右肩の検索窓で捜す）と、UIBasics-Controllers クラスのなかに見つかる。
そこで *Hook という文字列でメソッド検索する（このクラスを選んだ状態でブラウザの Find.. メニューの中にあるMethod.. を実行する）と、
このクラスの中に keyboardHook: というメソッドがあることがわかる。



そうやって見つけた keyboardHook: はこれ。



なのでこれを使ってキー入力アクションに介入すればいいか。
なお引数は aBlock となっており、フックルーチンをブロック（クロージャ）で与えるらしい。
そこでその通り、
postBuildWith: aBuilder

```
super postBuildWith: aBuilder.
(aBuilder componentAt: #codeText)
    ifNotNil:
        [:aWrapper |
         aWrapper widget controller keyboardHook: [ :aKeyboardEvent :aController | xxxxxx ] ]
```

としてみる。つまり
aWrapper widget controller
で codeText のコントローラである TextEditController が得られるので、それに向かって keyboardHook: と言いつける。
ただし引数はクロージャで、またそこには引数が二つあって（後述）、処理は xxxxx （今は適当）と。

試しに文字を取りだしてみる。
postBuildWith: aBuilder

```
super postBuildWith: aBuilder.
(aBuilder componentAt: #codeText)
    ifNotNil:
        [:aWrapper |
        aWrapper widget controller keyboardHook:
```

```
[:aKeyboardEvent :aController |
Transcript
    cr;
    show: aKeyboardEvent keyValue printString.
aKeyboardEvent yourself]]
```

処理としては、引数として渡ってきた aKeyboardEvent に対して keyValue と叫び、それを Transcript に出力する、だけ。
最後に自分自身をもう一度返しておかないといけない点に注意。（ aKeyboardEvent yourself がそれ）

実際にこれで実行して、codeText フィールドにタイプすると（下図では abc とタイプした）

Transcript には以下（ハイライト箇所）のように出る。

```
OldFashionedBrowser>>preBuildWith:
$a "16r0061"
$b "16r0062"
$c "16r0063"
```

ところで keyboardHook の引数は単に個性のない aBlock だった。
ここからどうやってそのブロックがどんな引数をもっていてくれるか、は、かなりわかりにくい。
今回はたまたま prependKeyboardHook というメソッドがすぐ近くにあって（上の図の keyboardHook: の 5 行下）、
そこを見ると一つの使用例を見ることが出来る。（prepend だから「その前に」かな）

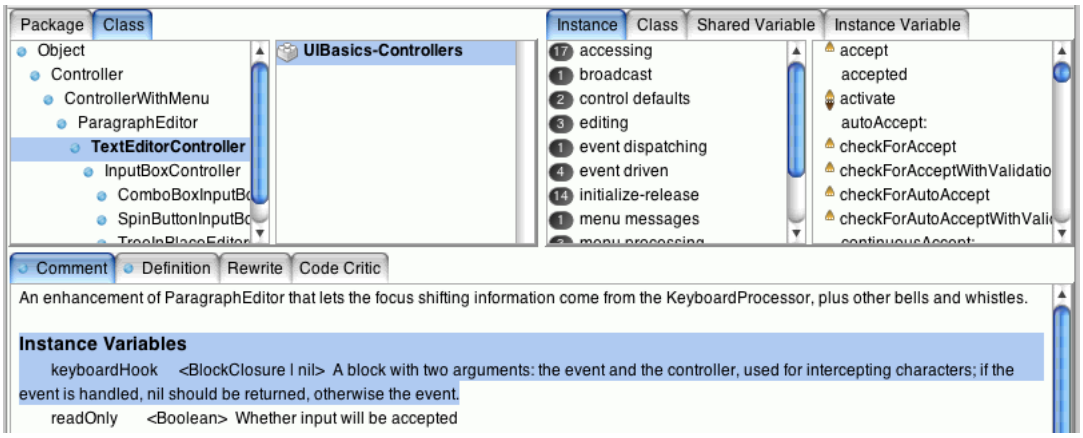
prependKeyboardHook: aBlock

"Install the supplied keyboard hook block, preserving the existing keyboard hook, 今のフックを残して（その前に）フック処理のブロックを入れる
if it exists. The new hook will be run first, then the old hook. If the new hook 新しいフックの方が先に走る。新しいフックが nil を返したら
returns nil, the old hook will not run. Sending this message multiple times それ以降の古いフックは実行しない。このメッセージは複数回送られる
establishes a chain of keyboard hooks run in sequence starting with the most ... あたりからちょっと意味不明。
recently set one."

```
| originalHook |
keyboardHook isNil ifTrue: [^self keyboardHook: aBlock]. 現在のフック（keyboardHook はインスタンス変数）が空なら与えられたブロックをバインド
originalHook := keyboardHook.
self keyboardHook: << あっ、keyboardHook: を使ってる（サンプルだ！）
    [:event :controller | << あっ、引数二つあった
    (aBlock value: event value: controller) ifNotNil:
        [:processedEvent | originalHook value: processedEvent value: controller]]
```

こういったものを発見しない限り判らないのではないか、というのが青木さんのコメント。
（まあこのメソッドにしても keyboardHook を中で使っているので、そこから捜してくるのだろうとは言え、、ちょっと厄介、、、）

あ、たまたまだけど、UIBasics-controller クラスのコメントとして、以下のような記述があった。（下部のハイライト箇所）



一応「二引数でイベントとコントローラ」とある。
まあしかし「あるところにはきっとあるが、どこにあるかは判らない」のパターンから脱していないのは確かだなあ。

また、ブロックには値があるが、これの仕様が何かをどうやって調べるか？という話もある。たぶんわかりにくい。

（この keyboardHook: が aBlock の値として自分自身のキーイベントを返すのについては、結局キーボードのフックルーチンなんだから、
フックして何かしらキーイベントを書き換える（例えば入力したアルファベットを大文字に変化させるとか）なりして返してくれよ、
その後のキーイベント処理に渡して処理してもらう（例えば TextEdit が入力文字をフィールドに格納する）からさ、という話なんだろう
なあと想像。）

これでキー入力をフックすることはできた。
次にメニューの項目（accept）を disable から enable にするような処理を作る。
先に示したように、これは

「ウィンドウのフックルーチンの中に、『キーイベントのフックルーチンを登録する処理』を登録する」
処理となる。
まずウィンドウの（ビルダーの）フックルーチン内で、所定の menu item を手に入れ、初期的に disable にするところから始める。
（後ろ三行の記述に注目）

```
postBuildWith: aBuilder
	| aMenu anItem |
	super postBuildWith: aBuilder.
	(aBuilder componentAt: #codeText)
		ifNotNil:
			[:aWrapper |
			aWrapper widget controller keyboardHook:
					[:aKeyboardEvent :aController |
					Transcript
						cr;
						show: aKeyboardEvent keyValue printString.
					aKeyboardEvent yourself]].
	aMenu := self menuAt: #codeTextMenu ifAbsent: [^nil].
	anItem := aMenu menuItemWithValue: #acceptCodeText.
	anItem disable
```

（単純にこのタイミングで上記コードを accept しようとすると menuAt:ifAbsent: がないと言われる。その場合は proceed すべし。
但し menuAt: （一引数）はあることに注意。menuAt:ifAbsent: はそれを拡張してこれから作るものである。直下を見よ。）

この三行とセットで private なインスタンスに、こんなのを作る。

```
menuAt: aSelector ifAbsent: aBlock
	^self builder
		ifNil: [self class perform: aSelector]
		ifNotNil: [:aBuilder | (aBuilder menuAt: aSelector) ifNil: [aBlock value]]
```

これは単に青木式のパターンらしい。

```
	^self builder
		ifNil: [self class perform: aSelector] << ビルダーが無ければクラスに投げる。きっと知っている。
		ifNotNil: [:aBuilder | (aBuilder menuAt: aSelector) ifNil: [aBlock value]] << あればそのビルダーに向かって menuAt: しなさい。
				もしそれがエラーを返すようなら（存在しなければ）、ifAbsent に書かれた代替処理用のブロックを実行して値を返せ。
```

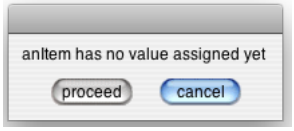
で、キー入力があったら enable にするだけなので、

```
	anItem enable
```

を、キー入力フックのクロージャに入れればよい。

```
postBuildWith: aBuilder
	| aMenu anItem |
	super postBuildWith: aBuilder.
	(aBuilder componentAt: #codeText)
		ifNotNil:
			[:aWrapper |
			aWrapper widget controller keyboardHook:
					[:aKeyboardEvent :aController |
					Transcript
						cr;
						show: aKeyboardEvent keyValue printString.
					anItem enable.
					aKeyboardEvent yourself]].
	aMenu := self menuAt: #codeTextMenu ifAbsent: [^nil].
	anItem := aMenu menuItemWithValue: #acceptCodeText.
	anItem disable
```

ただしこのまま accept しようとすると、以下のようなエラーが出る。



これは anItem がプログラム中の初出だから。実際のクロージャ実行時には下側にある

```
	anItem := aMenu menuItemWithValue: #acceptCodeText.
```

を通過しているので、問題なく処理されるのだが、コンパイラはそこを意識していない。
対応としては、下記のように前に持ってきても良い。クロージャ自体は後で評価されるから。

```
postBuildWith: aBuilder
	| aMenu anItem |
	super postBuildWith: aBuilder.
	aMenu := self menuAt: #codeTextMenu ifAbsent: [^nil].
	anItem := aMenu menuItemWithValue: #acceptCodeText.
	anItem disable.
	(aBuilder componentAt: #codeText)
		ifNotNil:
			[:aWrapper |
			aWrapper widget controller keyboardHook:
```

```
                    [:aKeyboardEvent :aController |
                    Transcript
                        cr;
                        show: aKeyboardEvent keyValue printString.
                    anItem enable.
                    aKeyboardEvent yourself]]
```

ところで
```
    aMenu := self menuAt: #codeTextMenu ifAbsent: [^nil].
```
の行は、
```
    aMenu := aBuilder menuAt: #codeTextMenu.
```
だけでもいい。
こうすると、menuAt:ifAbsent: を作らなくて良い。
単に menuAt:ifAbsent: は汎用性のためだけに作ったから、builder を得られているこの場では特別必要無い。

==========

勉強会ではaccept メニューを実行したときに acceptCodeText: from: メソッド（二引数）を呼び出すことになっているが、
僕は呼び出し（callback）メッセージを acceptCodeText: だけにしている。（ここの操作は追いついていなかった）
そのため僕は from: ナシでやっている。（★from:抜け、とマークしたところを参照）

次回、ここのところで矛盾が出るかも知れない。