今日は、始めに Serge(セルジュ)さんの Amber と Pharo のプレゼンとデモから
(英語力が壊滅的なので、大幅に間違ってるかもしれないので、話半分で…)

esug.org に所属してるって(European Smalltalker Users Group)
Amber Web ブラウザで動く Smalltalk
Web ページ埋め込み Smtalltalk
Javascript 使って動かすよ
コンパイルして、 Javascript バイトコードにする
Web アプリケーションみたいになる
Transcript も Workspace もある
SUnit (JUnit の Smalltalk 版だろう)もある
System Browser もある
halt とかもあるけど、動かない
JQuery と UI を Smalltalk を使って作れる？
Web ブラウザで、 Smalltalk で書いたプレゼンテーションだって出来るぜ！
 html を書いてる
修正して、保存して、再読込すればちゃんと変わる
スライドの中で Smalltalk プログラミングだって出来る
Smalltalk のオブジェクトと Javascript オブジェクトはそれぞれ対応している
イメージに相当する物はない？
セッションが切れると、オブジェクトは死ぬ？
コンパイルすると、 JSON 形式で、まるでオブジェクトな感じに化けてるように見
える
メソッドが無名関数に化けて、メッセージを投げるのもちゃんとやっている感じ
保存時にちゃんと、インスタンス変数か聞いてくる
気になるなら、オープンソースだから自分でダウンロードしてみてくれとのこと
Compiler クラスとか一見の価値あり
 ほとんど Smalltalk 部分で頑張っている

次に Pharo
オープンソースな Smalltalk プラットフォーム？
最新版は ver 1.3 ？
Mac, Windows で動くよ
git が使える？
git を含んでいる？

ここから、いつものー
USB メモリの中身

Fourier.tar.gz
SSK_20111102_205505.st
vw78jun791mac.zip
vw78jun79.win.zip

じゅんのバージョンが上がってる

SSK_20111102_205505.st を File in

ちょっと復習
Fourier1dModel, Class, examples, example1 を実行
Window の中でクリックすると Transcript に
SSK.PaneController>>redButtonPressedEvent: 20:10:30
などと出てくる

クリックして何かするようにしましょう
クリックすることによってインタラクションが居るのは powerSpectrumView だけなので

Fourier1dModel, Instance, aspects, sourceDataView を変更
sourceDataView

```
| fourierClass modelClass viewClass anImage aModel aView |
fourierClass := self defaultFourierTransformClass.
modelClass := self defaultPaneModelClass.
viewClass := self defaultPaneViewClass.
anImage := sourceData
                ifNil: [nil]
                ifNotNil: [fourierClass generateImageForData: sourceData].
aModel := modelClass picture: anImage.
aModel label: 'Source Data'.
aView := viewClass model: aModel.
aView alignmentSymbol: #center.
aView controller: NoController new.
^aView
```

NoController は何もしないコントローラ

これで、何も受け付けなくなる

同様に、 inverseDataView, realPartAndImaginaryPartView も NoController を追加する

Fourier1dModel, Instance, aspects, realPartAndImaginaryPartView を変更
realPartAndImaginaryPartView

```
	| fourierClass modelClass viewClass anImage aModel aView |
	fourierClass := self defaultFourierTransformClass.
	modelClass := self defaultPaneModelClass.
	viewClass := self defaultPaneViewClass.
	anImage := (realPart isNil or: [imaginaryPart isNil])
				ifTrue: [nil]
				ifFalse:
					[fourierClass generateImageForRealPart: (fourierClass
swap: realPart)
						andImaginaryPart: (fourierClass swap:
imaginaryPart)].
	aModel := modelClass picture: anImage.
	aModel label: 'Real Part and Imaginary Part'.
	aView := viewClass model: aModel.
	aView alignmentSymbol: #center.
	aView controller: NoController new.
	^aView
```

Fourier1dModel, Instance, aspects, inverseDataView を変更

inverseDataView

```
	| fourierClass modelClass viewClass anImage aModel aView |
	fourierClass := self defaultFourierTransformClass.
	modelClass := self defaultPaneModelClass.
	viewClass := self defaultPaneViewClass.
	anImage := inverseData
				ifNil: [nil]
				ifNotNil: [fourierClass generateImageForData: inverseData].
	aModel := modelClass picture: anImage.
	aModel label: 'Inverse Data'.
	aView := viewClass model: aModel.
	aView alignmentSymbol: #center.
	aView controller: NoController new.
	^aView
```

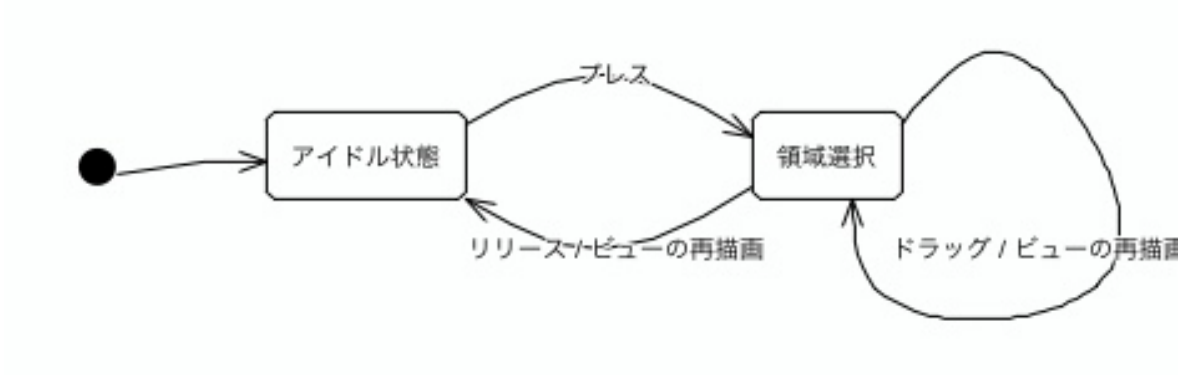
Controller の中に View の処理を入れてしまったりすることがあるが…

Smalltalk で、 View と Controller ははっきりと分かれているのはこのため

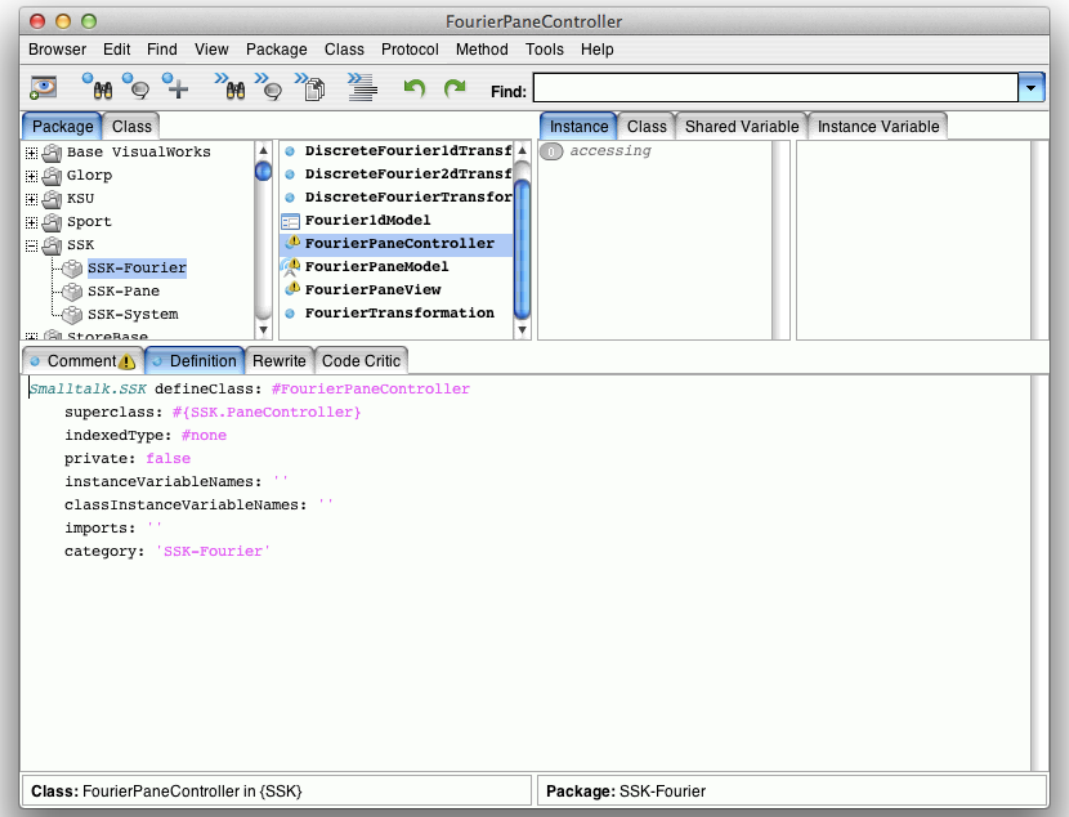
(Controller を差し替えられるようにするため)

これからやることは、 Power Spectrum の部分で、ドラッグによって、領域を選択したい

こういう遷移図になる



SSK-Fourier に FourierPane{Model, View, Controller} を追加
(これ以降 Comment をちゃんと編集していないために Warning マーク △の中に！があるが気にしないこと…)
SSK-Fourier の適当な Class の Definition を開いて、



Smalltalk.SSK defineClass: #FourierPaneController
superclass: #{SSK.PaneController}
indexedType: #none
private: false
instanceVariableNames: ''
classInstanceVariableNames: ''
imports: ''
category: 'SSK-Fourier'

このように変更して、 Accept

Model, View に関しても同様に行う

```
Smalltalk.SSK defineClass: #FourierPaneModel
	superclass: #{SSK.PaneModel}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'SSK-Fourier'

Smalltalk.SSK defineClass: #FourierPaneView
	superclass: #{SSK.PaneView}
	indexedType: #none
	private: false
	instanceVariableNames: ''
	classInstanceVariableNames: ''
	imports: ''
	category: 'SSK-Fourier'
```

FourierPaneView, Instance, controller accessing, defaultControllerClass を追加

```
defaultControllerClass

	^SSK.FourierPaneController
```

Fourier1dModel, Instance, aspects, powerSpectrumView を編集

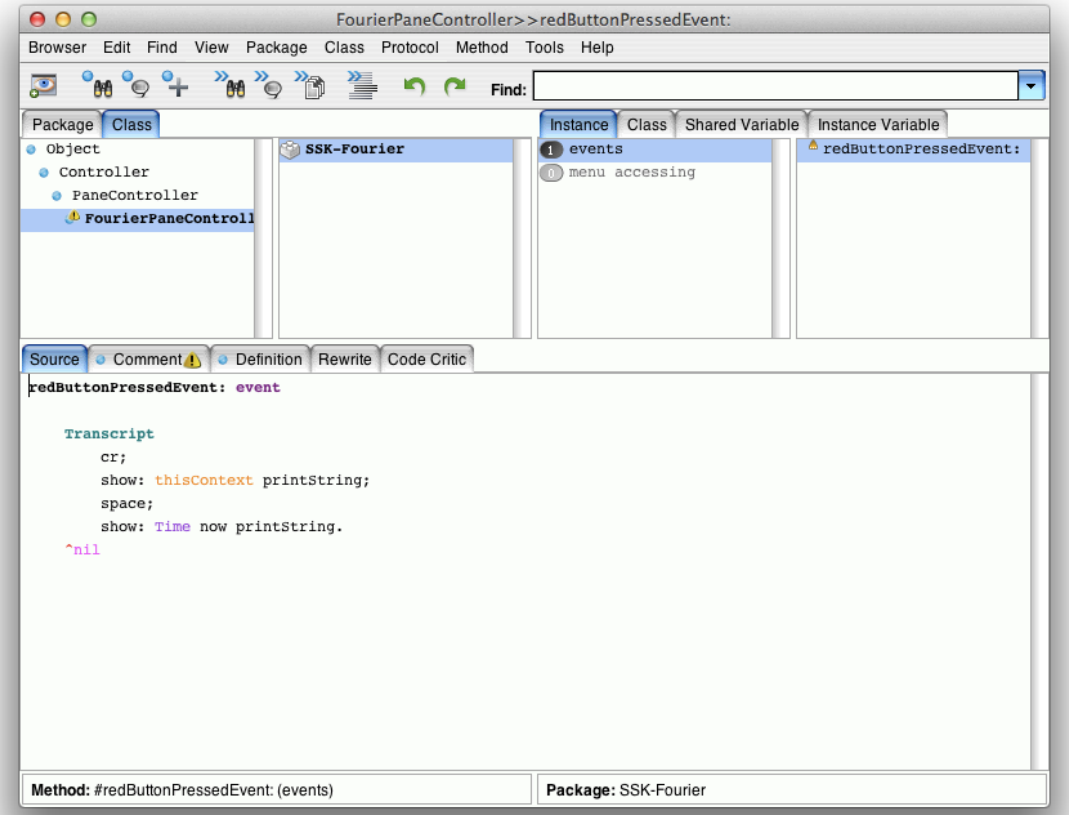
```
powerSpectrumView

	| fourierClass modelClass viewClass anImage aModel aView |
	fourierClass := self defaultFourierTransformClass.
	modelClass := FourierPaneModel.
	viewClass := FourierPaneView.
	anImage := powerSpectrum
			ifNil: [nil]
			ifNotNil:
				[fourierClass generateImageForSpectrum: (fourierClass
swap: powerSpectrum)].
	aModel := modelClass picture: anImage.
	aModel label: 'Power Spectrum'.
	aView := viewClass model: aModel.
	aView alignmentSymbol: #center.
	^aView
```

これで、先ほど作成した Model と View を使うようになる

とりあえず、 クリック時のイベントがとれるか確認

FourierPaneController, Instance, events, redButonPressedEvent を作成



redButtonPressedEvent: event

```
	Transcript
		cr;
		show: thisContext printString;
		space;
		show: Time now printString.
	^nil
```

と、入力して、 Accept
(この時点では差はない)

FourierPaneController, Instance, events, redButonPressedEvent を改めて編集
redButtonPressedEvent: event

```
	[self sensor redButtonPressed] whileTrue: [Processor yield].
	^nil
```

これがドラッグ中に何かをするときの基本の形
redButtonPressed ボタンが押されている間を意味する
yield 入力待ち中にイベントがない限りほかに CPU を優先する用に yield

中身を書いていく

```
redButtonPressedEvent: event

	| startingPoint previousPoint currentPoint |
	startingPoint := self sensor cursorPoint.
	previousPoint := startingPoint.
	[self sensor redButtonPressed] whileTrue:
		[currentPoint := self sensor cursorPoint.
		currentPoint = previousPoint
			ifFalse:
				[| aRectangle |
				aRectangle := Rectangle vertex: startingPoint vertex: currentPoint.
				Transcript
					cr;
					show: aRectangle printString].
		Processor yield].
	^nil
```

青色部分を見ると分かるようにボタンを押し込んでも、カーソルを動かしていない場合(ドラッグにはなっていない場合)は何も起こらない
が、カーソルを動かすと(押し込んでいる間 previousPoint は更新されないので)ドラッグしてる間 Transcript に、座標がだらーーーーっと出てくる

例えば、こんな感じ

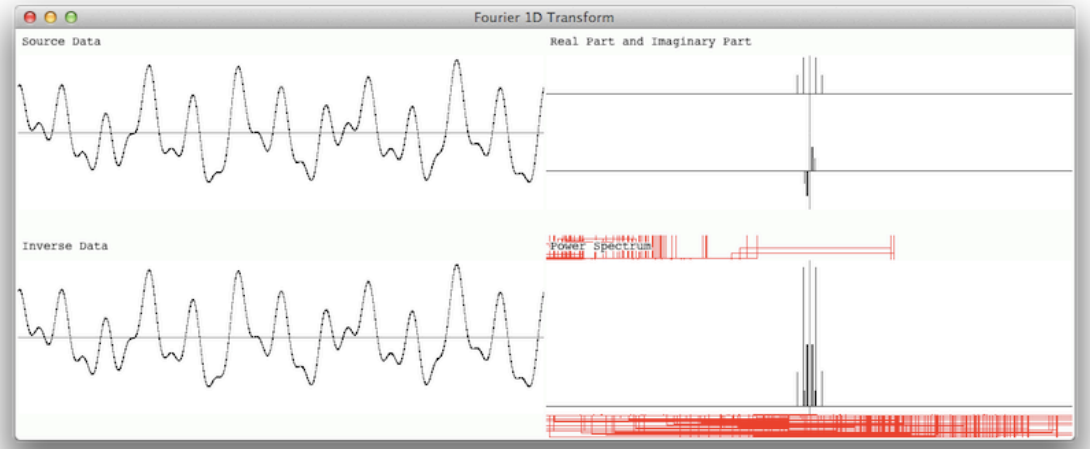
```
138 @ 84 corner: 191 @ 108
138 @ 84 corner: 191 @ 108
138 @ 84 corner: 180 @ 92
```

動いていることが確認できたところで、目的に合うように編集

```
redButtonPressedEvent: event

	| startingPoint previousPoint currentPoint graphicsContext |
	startingPoint := self sensor cursorPoint.
	previousPoint := startingPoint.
	graphicsContext := self view graphicsContext.
	[self sensor redButtonPressed] whileTrue:
		[currentPoint := self sensor cursorPoint.
		currentPoint = previousPoint
			ifFalse:
				[| aRectangle |
				self view displayOn: graphicsContext.
				aRectangle := Rectangle vertex: startingPoint vertex: currentPoint.
				graphicsContext
					paint: ColorValue red;
					displayRectangularBorder: aRectangle.
				previousPoint := currentPoint].
```

```
                Processor yield].
        self view displayOn: graphicsContext.
        ^nil
```



赤い線の残骸が残ってる

(Transcript が残っていて、そもそも良い物ではないので)とりあえず、修正
SSK-Pane, PaneView, Instance, displaying, displayOn: を修正
displayOn: graphicsContext

```
        Transcript
                cr;
                show: self bounds printString.
        self clearInside.
        self model picture
                ifNotNil:
                        [:anImage |
                        | messageSelector scaledImage aRectangle |
                        messageSelector := self alignmentSymbol.
                        scaledImage := self isScaling
                                        ifTrue: [anImage shrunkenBy: self scalingFactor
reciprocal]
                                        ifFalse: [anImage yourself].
                        aRectangle := scaledImage bounds.
                        aRectangle := aRectangle align: (aRectangle perform:
messageSelector)
                                        with: (self bounds perform: messageSelector).
                        scaledImage displayOn: graphicsContext at: aRectangle origin].
        self model label
                ifNotNil:
                        [:aString |
                        | aComposedText aPoint |
                        aComposedText := aString asComposedText.
                        aPoint := 4 @ 0.
                        graphicsContext paint: ColorValue white.
```

```
                (-1 to: 1) do:
                        [:y |
                        (-1 to: 1)
                                do: [:x | aComposedText displayOn:
graphicsContext at: aPoint + (x @ y)]].
                graphicsContext paint: ColorValue black.
                aComposedText displayOn: graphicsContext at: aPoint]
```

次はドラッグしてる間に、ちらちらするようになった

今回はここで時間切れ