

□ 2010.4.7

ProjectZ.zip は hitokuwomamorukoto で開く。

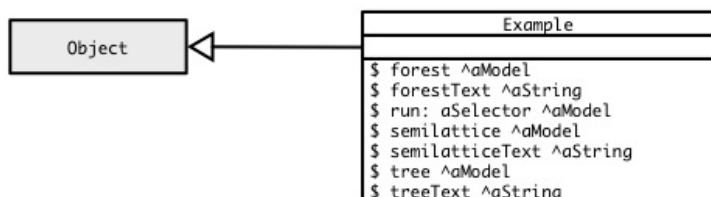
展開して、ProjectZ/Manual/index.html を見ると、そこに実行手順などあり。

要はこれ Project 演習のものだな。

これは Ruby, Objective-C, Java でも作っているらしい。が、Java, Smalltalk は動いたが、Ruby, Objective-C については OS 依存性を最後まででは取り除けなかったと言うてる。へええ。

・詳細設計のところ

以下の図があるが、



この \$ は Smalltalk という Class method に相当する、という意味。

他の言語であれば constructor に相当するか。

^ は戻り値の型を示す。

・テスト

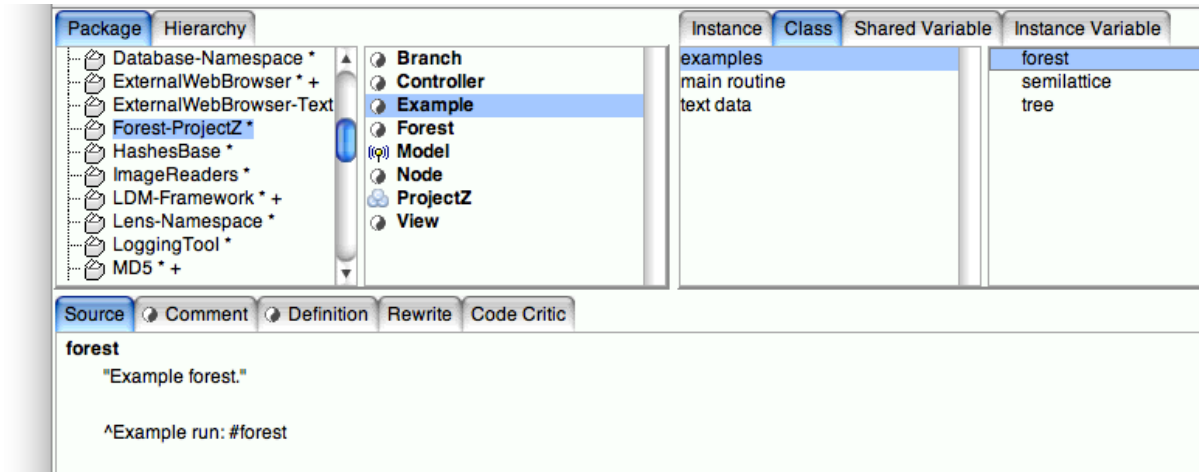
SUnit というのがあるらしい。つまり Smalltalk のユニットテストツール。

条件文と正しい結果を入れて、片っ端からメソッド呼び出しの結果の正しさを確認する。

・プログラムの中身を見る

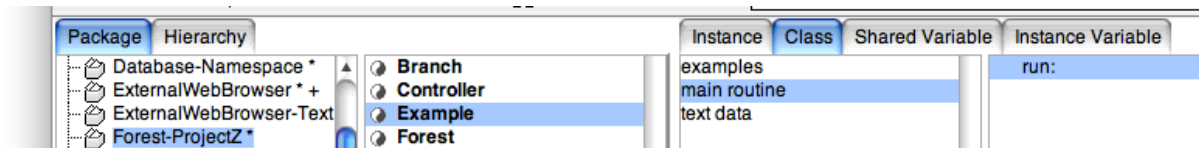
まず SharedVariable (大域変数)

次 Example



ここを ^self run:#forest と書かないのが青木方針、らしい。その場ではそれで動くけれども、問題はワークスペースに持って行った時に分からなくなるから。

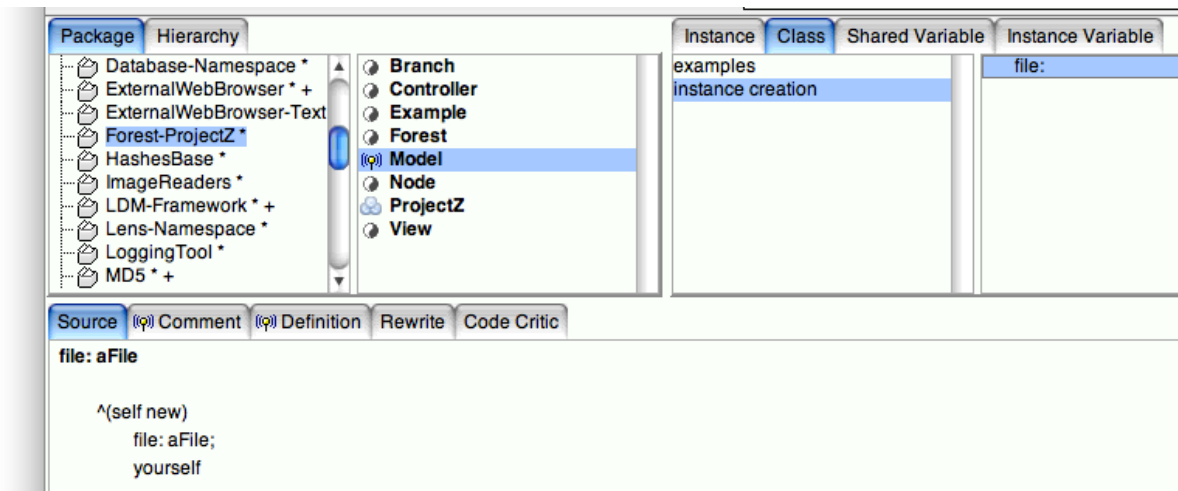
次、main routine（なんちゅう名前や）



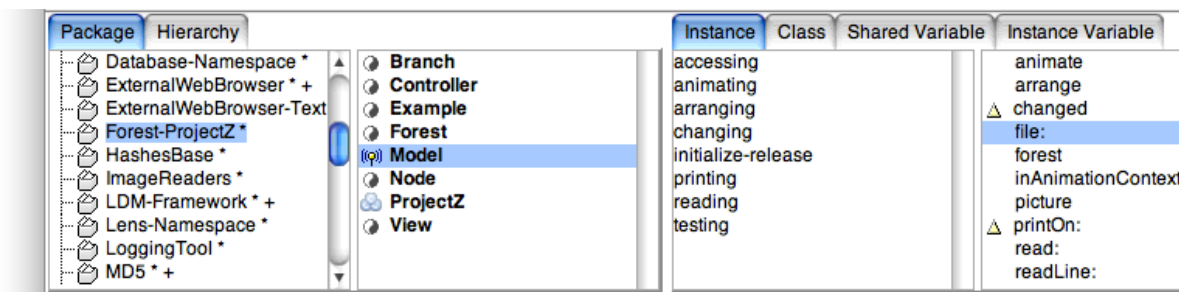
中身（ソース）は、

```
| aFile aModel aView aWindow aRectangle |
aFile := self perform: (aSelector, 'Text') asSymbol. ファイル名を決めて perform: につけて、
aModel := Model file: aFile. モデルを作って（ここにファイルオブジェクトを渡している、なんで file を渡す？）
aView := View model: aModel. View を作って（ここにモデルを渡している）
aWindow := ApplicationWindow ウィンドウを（これに aModel を渡す）
    model: aModel
    label: aFile tail asString
    minimumSize: 400 @ 300.
aWindow component: aView. 次に aView を教える
aRectangle := 0 @ 0 extent: 800 @ 600. 矩形をいろいろ作って、、、
aRectangle := aRectangle
    align: aRectangle center
    with: Screen default bounds center.
aWindow openIn: aRectangle. Window に矩形を指定して開けという
"aModel animate." んでアニメートせよ、と言う、、
                    というかこれはコメントだから、これを作れと青木さんは言っている？
^aModel 最後はモデルを返す
```

ところで設計書を見ると Model には Image が割り当てられるはずだが、それが上に見えない。
 どこだ？と思うと、ソレは結局 aModel := Model file: aFile. でしか無さそう。つまり file: メソッドを見る、と。
 で、見ると、



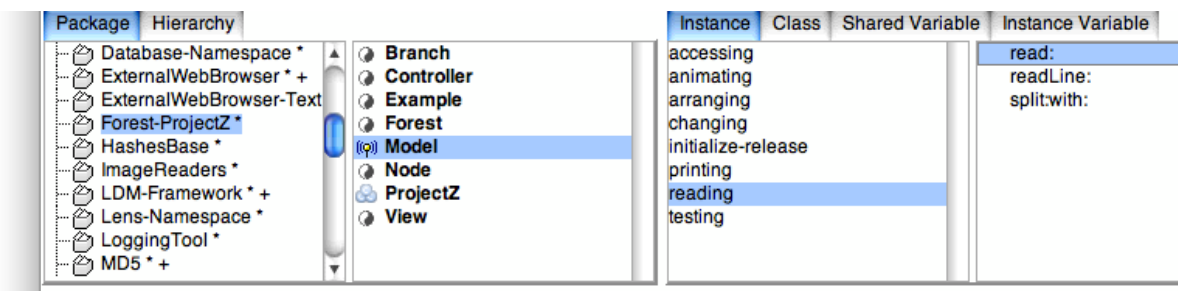
中身ないじゃん。クラスで受けてインスタンスを作って、それに対して同じ file: メッセージを投げつけている。
同じく aFile を渡して。
というわけでインスタンスを見る。



中身を見ると、

```
forest := Forest new.      Forestインスタンス（樹状整列の実体）を作って
picture := nil.            ピクチャーはまず空
aFile
    ifNotNil:
        [self read: aFile.   で、ファイルを read して
         self arrange]       何かアレンジするのね？
```

で、次に Forest には行かず read: に行く。深さ優先でやらない方が良く、というのが青木さんの教え。
まずインスタンスの read: へ行く。Forest new: へは後で戻る (★forest)



ソースは、

```
read: aFile
    | trees nodes branches aStream aString nodeArray stringArray anIndex aNode fromNode toNode aBranch |
    trees := OrderedCollection new.
    nodes := OrderedCollection new.
    branches := OrderedCollection new.
    aStream := (aFile asFilename withEncoding: DefaultEncoding) readStream.
    aStream lineEndAuto. // 行末コードなんでもあり対応
    [[aString := self readLine: aStream) notNil]    一行読むのね
        whileTrue:
```

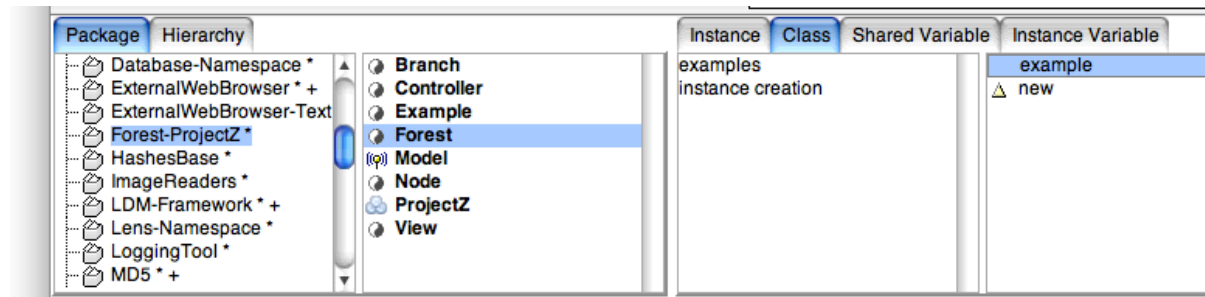
```

[aString = TagOfTrees      何か目印があったのか→実際には TagOfTrees は Constant にある。下の★1. 見て
  ifTrue:
    [[(aString := self readLine: aStream) notNil
      and: [(aString = TagOfNodes) not]] whileTrue: [trees add: aString]].
aString = TagOfNodes
  ifTrue:
    [[(aString := self readLine: aStream) notNil
      and: [(aString = TagOfBranches) not]] whileTrue: [nodes add: aString]].
aString = TagOfBranches
  ifTrue:
    [[(aString := self readLine: aStream) notNil
      whileTrue: [branches add: aString]]] ensure: [aStream close].
// で、ここまででファイルは全部メモリ上に取れた
nodeArray := Array new: nodes size.
nodes      ノードについては、
  do:       順繰りに
    [each | 片っ端から取り出して
      stringArray := self split: each with: ', '.      カンマ区切りで分けて、
      anIndex := stringArray first asNumber.            一番目は index として数値として解釈して取り出す
      aNode := Node named: stringArray last.            次はノード（グラフィック上の節点）（の名前）として取り出して
      nodeArray at: anIndex put: aNode.                 nodeArray に (anIndex => aNode) として突っ込んでいく
      forest addNode: aNode].                           んで forest にも突っ込んでおく（実体だけ）
branches    さて枝についても、
  do:       順繰りに
    [each | 片っ端から
      stringArray := self split: each with: ', '.      カンマ区切りで分けて、
      fromNode := nodeArray at: stringArray first asNumber.  入り口ノード番号を読んで、nodeArray からノード実体を
      toNode := nodeArray at: stringArray last asNumber.    出口ノード番号を取り、
      aBranch := Branch start: fromNode end: toNode.         枝をその両ノードから起こして、
      forest addBranch: aBranch].                           枝を登録する、だけ。いじょ。
// で、nodeArray はここで終わり。残さない。

```

★1. TagOfTrees などは text ファイル中に打ち込んであるマークに対応するもの。つまり入力ファイルにその種のマークがあったら、そこから先はこの分岐（モード）で読み続ける、という処理分岐をやりたい。

さて次は forest だ。（さっきの forest new のところ（★forest）に戻る）
example から見る。



ソースは以下の通り。
実行すると、小さなウィンドウを開いて、ブランチ一つ、ノード二つの最小図を描いてマウスに追随させるものを描く。

example

"Forest example."

```

| aForest fromNode toNode aBranch aWindow aSensor aGraphicsContext previousPoint currentPoint |
aForest := Forest new. ★1. あ、new だ
fromNode := Node named: 'Magnitude'.
toNode := Node named: 'Duration'.
aBranch := Branch start: fromNode end: toNode.
aForest
  addNode: fromNode;      ★2. なんかいりいり入れてる。入り口 node?
  addNode: toNode;        次の node?
  addBranch: aBranch.      枝、を入れるのかな？まあ枝は入り口と出口だから。      A→B だからね。（この→が枝）
aWindow := ApplicationWindow new.      んでウィンドウ開けるような雑処理か
aWindow openIn: (100 @ 100 extent: 400 @ 300).

```

```

aWindow displayPendingInvalidation.
InputState default cursorPoint: aWindow displayBox center.
aSensor := aWindow controller sensor.
aGraphicsContext := aWindow graphicsContext.
previousPoint := nil.
aSensor waitNoButton.                でボタンを待つと
[aSensor noButtonPressed]
  whileTrue:                          でボタンを押していない間、以下の作業をする
    [currentPoint := InputState default mousePoint.
     previousPoint = currentPoint      移動したら、
     ifFalse:
       [aGraphicsContext clear.        ここで描画。つまり aGraphicsContext を作るのね
        fromNode setLocation: aSensor cursorPoint.   ノードの位置設定（これは aForest の中に取り込まれてる）
        toNode setLocation: aSensor cursorPoint + (150 @ 100). もう一つのノードの位置設定（同上）
        aForest draw: aGraphicsContext.    ★3. まあとりあえず draw: と言えば描くのか、と
        previousPoint := currentPoint].
     Processor yield].
aWindow controller close. れれれ？と思うかも知れないけど、そう、この example はマウスをクリックすると終了する
^aForest

```

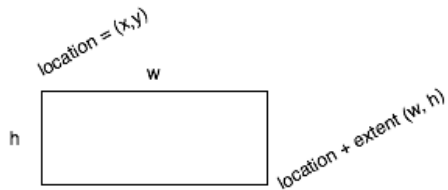
まず、

- ★1. aForest を new して、
 - ★2. それに向かってノードとブランチを登録し、
 - ★3. 再描画は aForest draw: と言えば描くのね、でそれをマウスに合わせてループしているのね？
- と読めばよい。（全体構造が分かる）

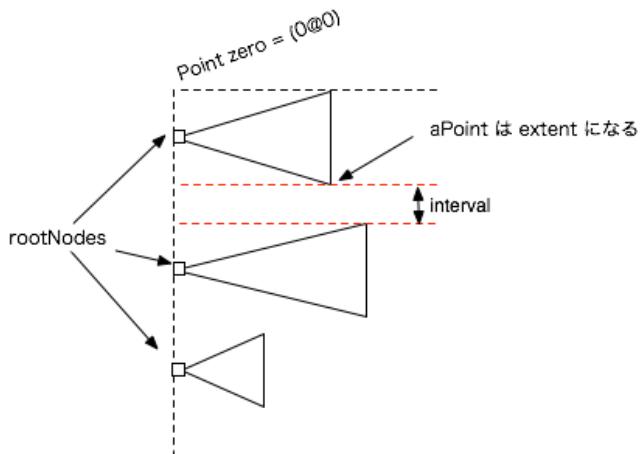
■ 5/12

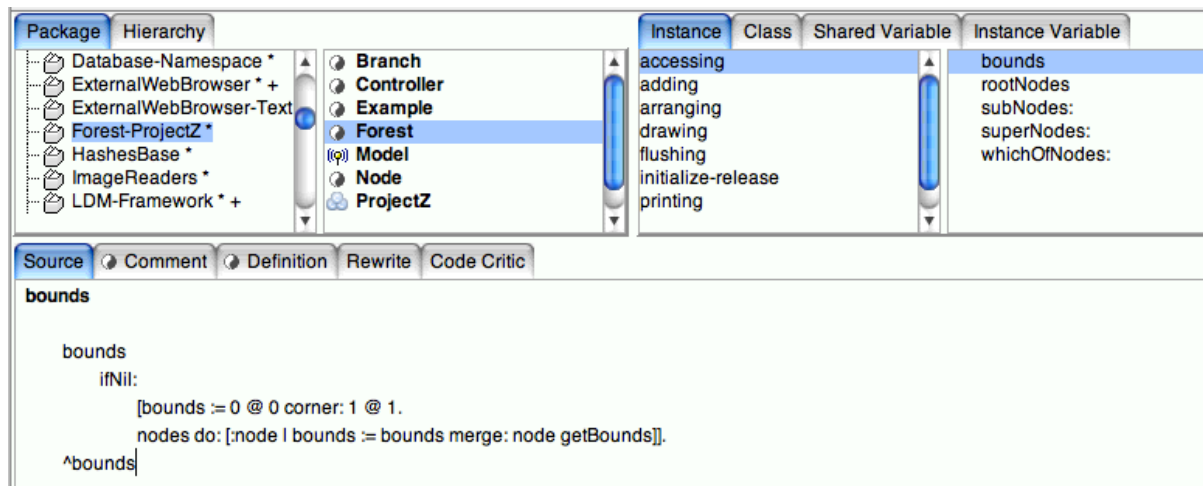
グラフィクス。

画像処理のために rectangle を扱う。location と extent , w, h の関係に注意。



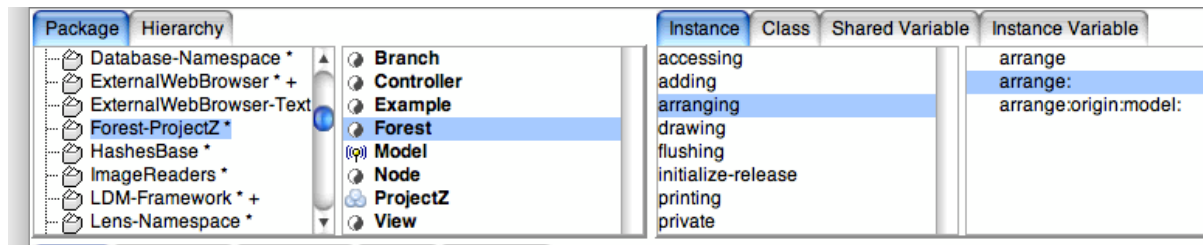
こんな感じに並べる。





merge は二つの rectangle をともに含む最小の rectangle を作る。
つまり bounds は bounds に新しく渡された node の bounds (getBounds で得る) との merge をとれば正しい新しい bounds が得られる。

次 arrange。複数のツリーを整理させる手続き。



コードは以下の通り。

```

arrange: aModel
| fontHeight yValue aPoint |
fontHeight := Node textAttributes lineGrid. << これでフォント情報が得られる
yValue := fontHeight + (Margin y * 2) + Interval y. << テキストの描画位置を Margin * 2 に隙間(2ピクセル) 離して描く
<< Margin についてはちょっと下の方に図を描いた
nodes << げげ、これなんじゃ。インスタンス変数か。キッとそれ以前に入れられてるんだな
with: (0 to: nodes size - 1)
do:
    [aNode :index |
    aNode << 片端からノードを設定する
        setStatus: UnVisited; << まず全部を unvisit である、というマークをつける (単に初期化と思えば良い)
        setLocation: 0 @ (yValue * index)]. << それらは左端の、縦一列に並べておく
    aPoint := Point zero. << 最初は aPoint は左上原点だよと
    self rootNodes << 自分自身に対して rootNodes を投げるとノード群の rootNode たちが得られる、 (直下に後述)
    do:
        [aNode | << それに含まれている親ノードを片端から一つとりだして、
        aPoint := self arrange: aNode origin: aPoint model: aModel. << aPoint を原点として親ノードから整列せえと言う
        aPoint := 0 @ (aPoint y + Interval y)]. << アレンジが終わったら戻ってきた y 位置 (一番下端位置) に Interval y ぶん下、
左端の場所を次の親ノード以下の整列開始位置とする
    self flushBounds

```

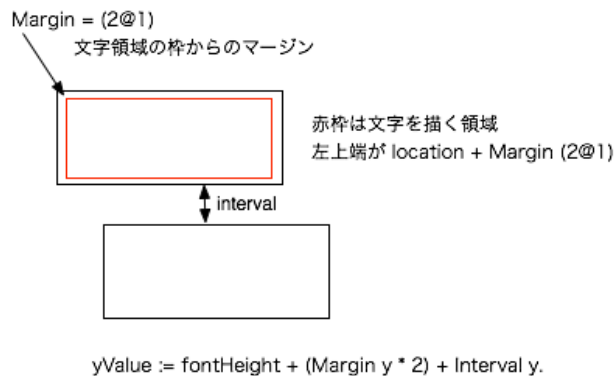
--- rootNodes メソッドだけ抜き出し

```

rootNodes
| rootNodes |
rootNodes := OrderedCollection new. << 順序つき配列を起こして、
nodes << げげ、これなんじゃ。インスタンス変数か。キッとそれ以前に入れられてるんだな
do:
    [aNode | (self superNodes: aNode) isEmpty ifTrue: [rootNodes add: aNode]]. << 親ノードがないノードは
rootNodes に追加
^self sortNodes: rootNodes << 一応ソートして返す

```

--- おわり



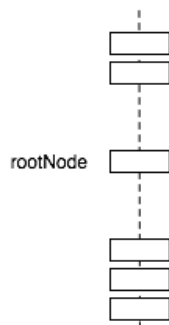
次 arrange: origin: model:
こいつは再帰的に呼ばれる。

```
arrange: aNode origin: aPoint model: aModel
| extent subNodes width height x y top h |
aNode setStatus: Visited. << そのノードは visited にマークする
aNode setLocation: aPoint.
self propagate: aModel. << 何か変化が起きたので再描画せえ、と言う。ここで aModel を利用
extent := aNode getExtent.
subNodes := self subNodes: aNode.
subNodes isEmpty
  ifTrue: << もしこのノードが leaf で tree でなければ、それ自身だけを結果の extent として返す
    [width := aPoint x + extent x.
     height := aPoint y + extent y.
     extent := width @ height.
     ^extent].
width := aPoint x + extent x. << そうでなければ (ツリーであれば) 全体 bounds の width を extent ぶん広げて
height := aPoint y. << 縦は単に y 位置のまま
x := width + Interval x. << 単に右に interval x を空けて隣の位置を新しい x とする
y := top := height.
subNodes
  do:
    [subNode |
     subNode getStatus == UnVisited
       ifTrue: << まだ visit してなければ (なぜこれが必要なんだ？計算量軽減？)
         [extent := self << ===== ここ再帰！！
          arrange: subNode
          origin: x @ y
          model: aModel.
          h := y + subNode getExtent y.
          y := extent y max: h.
          width := extent x max: width.
          height := extent y max: height.
          y := y + Interval y]].
y := y - Interval y. << ここから三行ちょっと変。top 頂点ノードをサブノード列の bounds の (上下的) 真ん中に移動させている
h := aNode getExtent y.
y > (aPoint y + h)
  ifTrue:
    [y := top + ((y - top - h) / 2).
     aNode setLocation: aPoint x @ y.
     self propagate: aModel]. << ここで aModel が登場
height := height max: h.
extent := width @ height.
^extent
```

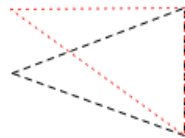
ノードってこんな感じ。

Node w
h

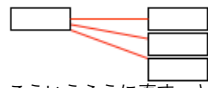
最初はノードは全部こんな感じで縦に並んでいるはず。そのなかの rootNode はここだと教えられる。



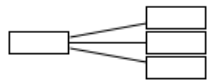
これを初めは赤枠のようなツリーに整列させ、後で黒枠のような位置に調整する。



一階層で見ると、こういう格好になっているものを、



こういうふうに直す、ということ。



僕は aModel がこのプログラムで主要な振る舞いをしているように見える（引数で再帰的に受け渡されている）のに、実際には何も仕事をしていない（整列操作に何の寄与もしていない）ことが非常に違和感があった。

結局その理由、引き渡さなければならないのは、つまり aModel を使っているのは propagate: による再描画のためだけ。

逆に aModel に描画させずに、self というか forest によって再描画させれば良いのだろうが、それを避けたい。

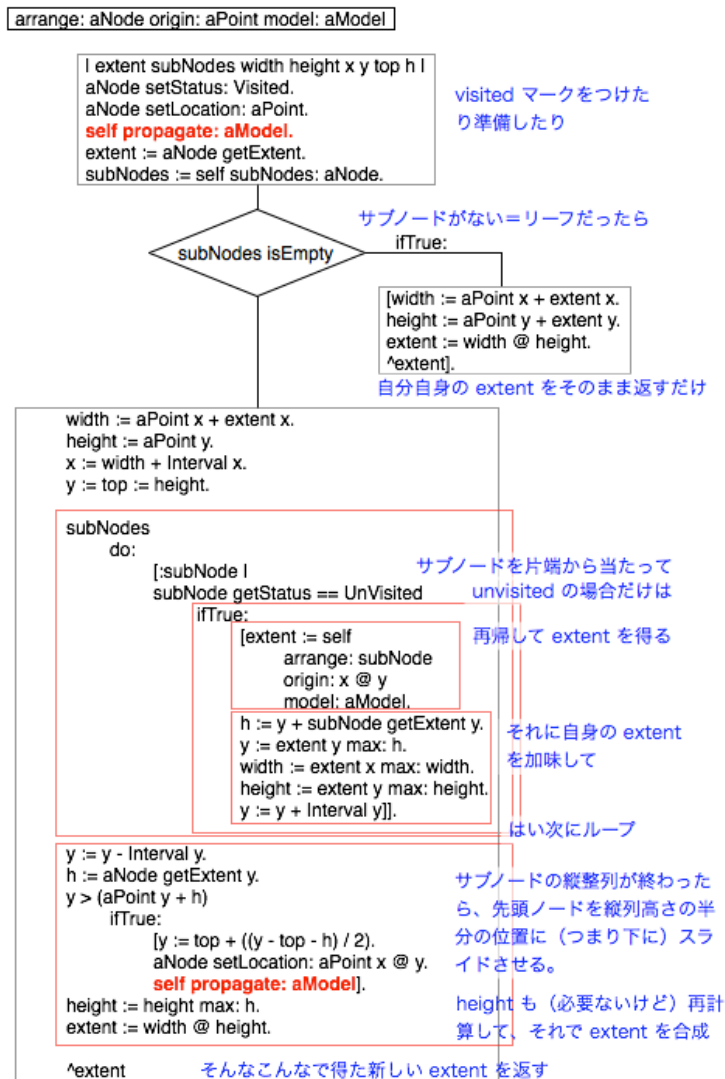
データ（forest）から画面処理(Model, MVC)を分離させたい。(MVC = observer パターンと言らしい)

が、困ったことにこれはアニメーションなので、整列を掛けている途中で observer である aModel が再描画してくれないと困る。

ストレートには forest に対して arrange（整列）しなさい、と言ったあとで、aModel からただ再描画しなさい、と言うだけで良いのだけれど、forest の arrange の中のループの途中で aModel に対して「再描画してくれ！」と言わなければならない。

Smalltalk ではこれをしようと思うと結局再帰呼び出しの引数に自分自身の model を引数として受け渡し続けて（教え続けて）やらないといけない。故にただ再描画のためだけに aModel を引数に与え続けることになる。

arrange プログラムの全体構造は以下の通り。



この図の中でやはり aModel が何もしていない（arrange=整列処理に何も寄与していない）ことが分かる。
（赤文字が aModel 利用箇所。なんとこれだけ。再描画せえと叫ぶだけ。）

もし forest が Smalltalk システムに対して notification を発することが出来て、それを受け取るように定義した aModel が非同期に実行する、などといった構造をとるのであればこうした引数による受け渡しの連鎖は不要と思うが、Smalltalk はそれ自身グローバルなイベントを待ち受けるような処理にはできていないので無理、ということか。

それにしてもこういう構造をそのまま見せてくれるエディタは無いのか？

何故か Smalltalk はこうした構造が見えてくるまで非常に非常に「まるごと分からない」という状態になる。

慣れの問題、というだけではないような。

青木さんは「キャストが分からないと、舞台が見えないと分からない」と言うが、そういう感じ。