

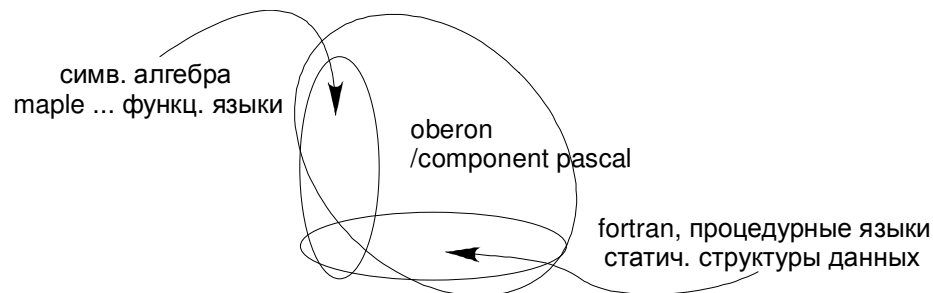
# С/к Введение в современное программирование (v.5.5)

Физфак МГУ. 2006/7 уч. год.

## Лекция 21

### Введение ко второму семестру

Практический смысл курса выражается картинкой:



Второго семестра раньше не было —> несколько хаотично.  
Ряд тем не закончено/отложено.

Перечислим произвольном порядке:

- 14 Продолжить рисование
- 15 Сортировки (с учетом алгебраических)
- 16 Генераторы комбинаторных объектов
- 17 Передовые способы работы с функциями
- 18 Некоторые важные вычислит. алгоритмы
- 19 ...

## Views — продолжение

### Напоминание

Модель, принятая в Оберонах/ББ, -- на поколение вперед .NET (там содрано с Борланда).

**Порт** (Port) -- абстракция изображающего устройства -- экран, принтер.

**Фрейм** (Frame) -- абстракция доступа к порту ("окошко").

**Вьюшка** (View) -- объект, умеющий себя рисовать.

Вьюшка создается программистом под задачу.

Вьюшка сама себя рисует (по сигналу ББ):

```
PROCEDURE ( v: View ) Restore ( f: Frame; l, t, r, b: INTEGER ), NEW, ABSTRACT;
```

рисует в "порт" через передаваемый ей "фрейм", используя штатные примитивы:

```
PROCEDURE ( f: Ports.Frame ) DrawLine ( x0, y0, x1, y1, s: INTEGER; col: Ports.Color ), NEW;
```

Вьюшка сообщает ББ о своих свойствах (в ответ на запрос ББ):

```
PROCEDURE ( v: View ) HandlePropMsg- ( VAR msg: PropMessage ), NEW, EMPTY;
```

37 Вьюшки можно открывать в окошке, спасать в файл и т.п.:

```
38 PROCEDURE OpenView ( v: View );
```

```
39 PROCEDURE RegisterView ( v: View; loc: Files.Locator; name: Files.Name );
```

```
40 PROCEDURE OldView ( loc: Files.Locator; name: Files.Name ) View;
```

41 вставлять в другие вьюшки:

```
42 PROCEDURE InstallFrame ( host: Frame; v: View; x, y, level: INTEGER; focus:
```

```
43 BOOLEAN );
```

44 Вьюшкам можно передавать сообщения, чтобы они как-то менялись:

```
45 PROCEDURE Omnicast ( VAR msg: ANYREC ); и т.п.
```

46

47 Были усложняющиеся примеры вьюшек:

48 сначала только рисовала одну линию;

49 потом научили ее сообщать ББ о своем размере;

50 потом научили ее хранить содержимое и рисовать в зависимости;

51 научились обновлять конкретную вьюшку "на лету" (Views.Update( v, ... ))

52 научили ее сохраняться и восстанавливаться из "файла" -- автоматически  
53 получили возможность копировать в Ворд-файл.

54

## Усложняем задачу. Простой график функции

56 Пусть есть функция (какой-нить синус).

57 Какие есть примитивы для рисования? Нужно смотреть Ports.Frame (Ctrl+D):

58 TYPE

```
59 Frame = POINTER TO ABSTRACT RECORD
```

```
60 unit-, dot-: INTEGER;
```

```
61 rider-: Ports.Rider;
```

```
62 gx-, gy-: INTEGER;
```

```
63 ( f: Frame ) CharIndex ( x, pos: INTEGER; IN s: ARRAY OF CHAR; font: Fonts.Font );
```

```
64 INTEGER, NEW;
```

```
65 ( f: Frame ) CharPos ( x, index: INTEGER; IN s: ARRAY OF CHAR; font: Fonts.Font );
```

```
66 INTEGER, NEW;
```

```
67 ( f: Frame ) ConnectTo ( p: Ports.Port ), NEW, EXTENSIBLE;
```

```
68 ( f: Frame ) DrawLine ( x0, y0, x1, y1, s: INTEGER; col: Ports.Color ), NEW;
```

```
69 ( f: Frame ) DrawOval ( l, t, r, b, s: INTEGER; col: Ports.Color), NEW;
```

```
70 ( f: Frame ) DrawPath ( IN p: ARRAY OF Ports.Point; n, s: INTEGER; col: Ports.Color;
```

```
71 path: INTEGER ), NEW;
```

```
72 ( f: Frame ) DrawRect ( l, t, r, b, s: INTEGER; col: Ports.Color ), NEW;
```

```
73 ( f: Frame ) DrawSSString ( x, y: INTEGER; col: Ports.Color; IN s: ARRAY OF
```

```
74 SHORTCHAR; font: Fonts.Font ), NEW;
```

```
75 ( f: Frame ) DrawString ( x, y: INTEGER; col: Ports.Color; IN s: ARRAY OF CHAR; font:
```

```
76 Fonts.Font ), NEW;
```

```
77 ( f: Frame ) Input ( OUT x, y: INTEGER; OUT modifiers: SET; OUT isDown: BOOLEAN ),
```

```
78 NEW;
```

```
79 ( f: Frame ) MarkRect ( l, t, r, b, s, mode: INTEGER; show: BOOLEAN ), NEW;
```

```
80 ( f: Frame ) RestoreRect ( l, t, r, b: INTEGER; dispose: BOOLEAN ), NEW;
```

```
81 ( f: Frame ) SCharIndex ( x, pos: INTEGER; IN s: ARRAY OF SHORTCHAR; font:
```

```
82 Fonts.Font ): INTEGER, NEW;
```

```
83 ( f: Frame ) SCharPos ( x, index: INTEGER; IN s: ARRAY OF SHORTCHAR; font:
```

```
84 Fonts.Font ): INTEGER, NEW;
```

```
85 ( f: Frame ) SaveRect ( l, t, r, b: INTEGER; VAR res: INTEGER ), NEW;
```

```
86 ( f: Frame ) Scroll ( dx, dy: INTEGER ), NEW;
```

```
87 ( f: Frame ) SetCursor ( cursor: INTEGER ), NEW;
```

```
88 ( f: Frame ) SetOffset ( gx, gy: INTEGER ), NEW, EXTENSIBLE
```

```
89 END;
```

90

```

91 Мы должны свести рисование функции к этим примитивам.
92 1) отдельные линии (уже умеем); 2) ломаные линии; 3) кривые Безье (самый
93 мощный).
94 1) неинтересно, 3) пока сложно. Выберем 2) — ломаные линии — DrawPath.
95
96 Быстренько проверяем, что там за типы параметров (Ctrl+D):
97 тип Ports.Point определен как RECORD x, y: INTEGER END;
98 Очевидно (логика ББ), это могут быть только координаты сегментов ломаной в
99 универсальных "экранных" координатах.
100
101 Итак, имея f, нужно приготовить последовательность точек, а затем передать ее
102 процедуре DrawPath.
103
104 Чтобы сэкономить время, заодно добавим и простое ...
105
106 Взаимодействие с клавиатурой
107 Хотим, чтобы при нажатии на +/- число точек интерполяции менялось на +/-1.
108 Вот модуль:
109
110 MODULE Kurs2006Graph;
111   IMPORT Log := StdLog, Math, Views, Ports, Stores, Properties, Controllers;
112
113   CONST mm = Ports.mm;
114         size = 50 * Ports.mm; aDefault = 0.0; nDefault = 4;
115
116   TYPE
117     View* = POINTER TO RECORD ( Views.View )
118       n: INTEGER;
119       a: REAL
120     END;
121
122   PROCEDURE Fun ( x: REAL ): REAL; (* рисуемая функция *)
123   BEGIN RETURN Math.Sin( 3 * x )
124   END Fun;
125
126   PROCEDURE ( v: View ) Restore* ( f: Views.Frame; l, t, r, b: INTEGER );
127   VAR p: POINTER TO ARRAY OF Ports.Point; scale, x, y: REAL; i: INTEGER;
128   BEGIN
129     scale := size;
130     (* приготовить последов. точек *)
131     NEW( p, v.n );
132     FOR i := 0 TO v.n - 1 DO
133       x := i / ( v.n - 1 );
134       y := Fun( x ) + v.a;
135       p[ i ].x := SHORT( ENTIER( scale * x ) );
136       p[ i ].y := SHORT( ENTIER( - scale * y ) ) + size
137     END;
138     (* нарисовать: *)
139     f.DrawRect( 0, 0, size, size, 0, Ports.green );
140     f.DrawPath( p, v.n, 0, Ports.black, Ports.openPoly )
141   END Restore;
142
143   PROCEDURE ( v: View ) Externalize- ( VAR wr: Stores.Writer );
144   BEGIN wr.WriteInt( v.n ); wr.WriteReal( v.a )
145   END Externalize;

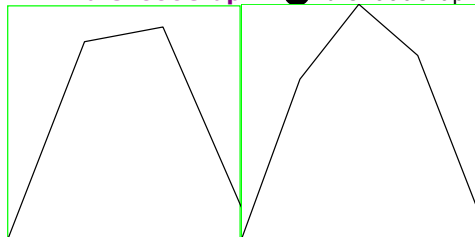
```

```

141 PROCEDURE ( v: View ) Internalize- ( VAR rd: Stores.Reader );
142 BEGIN rd.ReadInt( v.n ); rd.ReadReal( v.a )
143 END Internalize;
144
145 PROCEDURE ( v: View ) CopyFromSimpleView- ( source: Views.View );
146 BEGIN WITH source: View DO v.n := source.n; v.a := source.a END
147 END CopyFromSimpleView;
148
149 PROCEDURE ( v: View ) HandlePropMsg- ( VAR p: Properties.Message );
150 BEGIN
151   WITH p: Properties.SizePref DO
152     p.w := size; p.h := size
153   ELSE
154     END;
155   END HandlePropMsg;
156
157 PROCEDURE ( v: View ) HandleCtrlMsg* ( f: Views.Frame;
158   VAR msg: Controllers.Message;
159   VAR focus: Views.View );
160
161   CONST (* взятые из F1 --> Character set *)
162     aL = 1CX; (* arrow left key *)
163     aR = 1DX; (* arrow right key *)
164     aU = 1EX; (* arrow up key *)
165     aD = 1FX; (* arrow down key *)
166     esc = 1BX; (* escape key *)
167   VAR update: BOOLEAN;
168   BEGIN
169     WITH msg: Controllers.EditMsg DO
170       IF msg.op = Controllers.pasteChar THEN(* accept typing *)
171         update := TRUE;
172         CASE msg.char OF
173           "+" : INC( v.n )
174           "-" : v.n := MAX( v.n - 1, 2 )
175           "aU" : v.a := v.a + 0.1
176           "aD" : v.a := v.a - 0.1
177           |esc : v.n := nDefault; v.a := aDefault
178           ELSE
179             update := FALSE
180           END;
181       IF update THEN
182         Views.SetDirty( v ); (* mark view's document as dirty *)
183         Views.Update( v, Views.keepFrames ) (* restore v in any frame
184           that displays it *)
185       END
186     ELSE
187       (* ignore other messages *)
188     END
189   END HandleCtrlMsg;
190
191 PROCEDURE New*;
192   VAR v: View;
193   BEGIN NEW( v ); v.a := aDefault; v.n := nDefault; Views.OpenView( v )
194   END New;

```

194 END **Kurs2006Graph.** !Kurs2006Graph.New



195  
196 Первоначальная картинка -- слева.

197 Одно нажатие + (удобнее нажимать серые +/-) делает картинку справа -- на  
198 один узел больше.

199 Теперь нажимайте +/- и наслаждайтесь тем, как график становится  
200 глаже/грубее.

201 Нажимая стрелки вверх/вниз, видим, как меняется форма кривой.

202 Справа после двух нажатий на серый + и двух стрелок вниз.

203 **Даже скопировав в Ворд** — просто двойной щелчок там по картинке — и  
204 нажимайте стрелки!

#### 205 Упр

206 1. Сделать так, чтобы v.n сохранялось в файл/при копировании (см. лекцию 14).

207 2. Изображать график на произвольном сегменте по x с произвольными  
208 масштабами по x и y.

209 3. Рисовать несколько функций одновременно на одной картинке (например,  
210 разными цветами).

211 4. Рисовать в виде крестиков набор эксп. пар x, y; то же одновременно с  
212 некоторой функцией.

213 5. Для заданного набора данных Data, вычислить гистограмму с заданным  
214 числом бинов n и изобразить ее. Разрешить менять шаг гистограммы с помощью  
215 +/-.

216 6. Обдумать, как сделать шаг зависящим от крутизны функции (производная).

217 7. Обдумать, как рисовать график штриховой линией с заданной длиной штриха.

218 Пунктирной? Штрих-пунктирной?

219 8. Печатать в картинку значения v.a и v.n (Ports.Frame.DrawString).

#### 220 Печать

221 Ясно, что печатать иногда полезно.

222 Печать -- аналог рисования. Так ББ и трактует.

223 Т.е. вообще-то чтобы печатать, **ничего делать не надо.**

#### 224 Тонкость:

225 0) картинка может быть сложной:

226 на экране рисуем в грубом разрешении, печатаем в высоком (скажем, больше  
227 сегментов у ломаной и т.п.)

228 1) вьюшки иногда умеют растягиваться вслед за растягиванием окошка. Но  
229 "окошко" для печати -- это вся страница. Такая вьюшка будет хотеть  
230 нарисоваться в размер листа — может, это не то, чего мы хотим.

231 Для таких ситуаций есть процедура Ports.IsPrinterPort ( p: Port ): BOOLEAN;

232 Она как раз позволяет вьюшке определить, что в данном вызове Restore

233 "рисуемся" в принтер, и "рисоваться" по-другому (более тщательно или с

234 другим масштабом — не в полный размер листа).

#### 235 Создание картинок в других форматах

236 Это постоянно нужно при написании статей, для создания презентаций в  
237 специализир. программах, для "обогащения" картинки, порожденной ББ,  
238 спецэффектами (график на фоне фотографии ускорителя и т.п.).

#### 239 BMP и WMF

240 Просто Paste Special... и там будет меню для выбора.

#### 241 \*\*\*PDF

242 Очень важный формат, родственник PostScript'a, стандарт де факто -- принят,  
243 например, в Mac OS X в качестве основного механизма of rendering.

244 Можно взять бесплатную программку CutePDF Writer -- она работает как

245 драйвер для принтера: т.е. мы печатаем что-то как бы в принтер CutePDF Writer,  
246 а драйвер создает pdf (он спросит про имя файла).

247 После этого можно брать pdf-файл и делать с ним что-то.

#### 248 Другие форматы

249 Например, графический редактор Canvas X (ACD) тоже предоставляет  
250 аналогичный механизм: устанавливается некий драйвер как бы для принтера, но  
251 при печати в него рисуется соотв. рисунок в новом документе этой программы.

252 **Кстати:** ACD Canvas X ведет происхождение из мира Макинтошей (где  
253 процветает высококач. професс. графический дизайн) и является уникальной в  
254 том, что уже больше 10 лет позволяет смешивать в одном "произведении"  
255 пиксельную и векторную графику, а также и тексты. Замена сразу целой куче  
256 программ (Illustrator + Photoshop + PowerPoint). **Recommended.** (MS только  
257 еще делает такую.)

#### 259 Как делать движущуюся картинку

260 Сначала Actions: модуль Services

261

262 DEFINITION Services;

263 CONST immediately = -1; now = 0;  
264 resolution = 1000; кол-во тиков в секунде

265 TYPE

266 Action = POINTER TO ABSTRACT RECORD  
267 ( a: Action ) Do-, NEW, ABSTRACT выполняемые действия  
268 END;

269 PROCEDURE Ticks (): LONGINT; текущее время в тиках

270 PROCEDURE DoLater ( a: Action; notBefore: LONGINT ); поставить в очередь  
271 на выполнение

272 immediately = как часть текущей команды

273 now = немедленно после текущей команды

274 >0 = через сколько-то миллисекунд

275 **Action может само себя повторно поставить на выполнение =**  
276 основа всяких повторяющихся действий (эволюции)

277

278 PROCEDURE RemoveAction ( a: Action ); удалить из очереди на выполнение

279 ...

280 END Services.

281 **Упр** Взять потенциально длинный расчет (интегрирование по Монте Карло  
 282 сложной функции), разбить его на небольшие куски, вычисление куска  
 283 реализовать как Action.Do, текущий кусок пусть ставит очередной кусок (или  
 284 самого себя с другими параметрами) в очередь на выполнение с параметром  
 285 potBefore = pow или маленьким положит. значением -- напр., 1). Тогда ББ не  
 286 будет заблокирован на время счета.

287 **NB** Накладные расходы на действия с Action не пренебрежимы. Action.Do  
 288 должен делать существенный кусок вычислений (не одно суммирование из  
 289 миллиона, а десять тысяч... типа).

## 290 "Живой" визуальный объект

```
291 MODULE Kurs2006EvolvingView; (* adapted from ObxCubes *)
292   IMPORT Views, Ports, Properties, Services, Stores, Math;
293   CONST size = 20 * Ports.mm; sec = Services.resolution;
294   TYPE
295     View = POINTER TO RECORD ( Views.View )
296       phi: REAL (* угол наклона "стрелки" в радианах *)
297     END;
298   Action = POINTER TO RECORD ( Services.Action )
299     v: View;
300     next: Action;
301   END;
302   VAR inc: REAL; first: Action;
303   PROCEDURE ( a: Action ) Do;
304   BEGIN
305     (* "сложное вычисление" "за кадром": *)
306     a.v.phi := ( a.v.phi + inc );
307   Services.DoLater( a, Services.Ticks() + sec (* Services.now *) );
308   (* оставляем заказ Блэкбоксу обновить конкретную картинку v: *)
309   Views.Update( a.v, Views.keepFrames ); (* the update is performed
310   delayed *)
311   END Do;
312   PROCEDURE ( v: View ) HandlePropMsg ( VAR msg: Properties.Message );
313   BEGIN
314     WITH msg: Properties.SizePref DO
315       msg.w := size; msg.h := size
316     ELSE
317       END
318     END HandlePropMsg;
319   END;
```

```
324 PROCEDURE ( v: View ) Restore ( f: Views.Frame; l, t, r_, b: INTEGER );
325   VAR rad, x, y: INTEGER;
326   BEGIN
327     f.DrawOval( 0, 0, size, size, 0, Ports.red );
328     rad := size DIV 2;
329     x := SHORT( ENTIER( rad + rad * Math.Cos( v.phi ) ) );
330     y := SHORT( ENTIER( rad + rad * Math.Sin( v.phi ) ) );
331     f.DrawLine( rad, rad, x, y, 0, Ports.blue )
332   END Restore;
333   PROCEDURE Start ( a: Action );
334   BEGIN
335     a.next := first; first := a;
336     Services.DoLater( a, Services.now )
337   END Start;
338   PROCEDURE New*;
339   VAR a: Action;
340   BEGIN
341     NEW( a ); NEW( a.v ); a.v.phi := - Math.Pi() / 2;
342     Views.OpenView( a.v );
343     Start( a )
344   END New;
345   PROCEDURE Close; (* гигиена *)
346   BEGIN
347     WHILE first # NIL DO
348       Services.RemoveAction( first );
349       first := first.next
350     END;
351   END Close;
352   BEGIN
353     inc := Math.Pi() / 30
354   CLOSE
355   Close
356   END Kurs2006EvolvingView. ❶ Kurs2006EvolvingView.New (много раз)
357   NB Копируются OK, но не шевелятся -- и в начальном положении
358   Упр Почему? Как добиться, чтобы двигались все, причем копировалось бы и
359   состояние?
360   Упр Изменить так, чтобы показывалось не внутреннее состояние, а настоящее
361   время (секунды). Использовать средства модуля Dates (тип Dates.Time и
362   процедура Dates.GetTime).
```

## Простой пример работы с мышкой

```

370
371
372 MODULE Kurs2006Mouse;
373
374     IMPORT Stores, Ports, Models, Views, Controllers, Properties;
375
376     CONST minVersion = 0; maxVersion = 0;
377
378     TYPE
379         Line = POINTER TO RECORD (* линии организованы в виде списка *)
380             next: Line;
381             x0, y0, x1, y1: INTEGER;
382             c: Ports.Color
383         END;
384
385         View = POINTER TO RECORD ( Views.View ) (* вьюшка, которая рисует
386 линии *)
387             lines: Line
388         END;
389
390     VAR color: Ports.Color; (* текущий цвет для нового сегмента *)
391
392     PROCEDURE ( v: View ) Restore ( f: Views.Frame; l, t, r, b: INTEGER );
393     VAR p: Line;
394     BEGIN
395         (* проходим по списку и просто рисуем каждую линию: *)
396         p := v.lines;
397         WHILE p # NIL DO
398             f.DrawLine( p.x0, p.y0, p.x1, p.y1, f.dot, p.c );
399             p := p.next
400         END
401     END Restore;
402
403     PROCEDURE ( v: View ) Insert ( x0, y0, x1, y1: INTEGER ), NEW; (* добавляем
404 новую линию во вьюшку *)
405     VAR l: Line;
406     BEGIN
407         NEW( l );
408         l.x0 := x0; l.y0 := y0; l.x1 := x1; l.y1 := y1;
409         l.c := color;
410         (* обычная вставка в список: *)
411         l.next := v.lines; v.lines := l;
412     END Insert;
413
414     PROCEDURE GetBox ( x0, y0, x1, y1: INTEGER; OUT l, t, r, b: INTEGER ); (*
415 область вьюшки, выраженная станд. образом в терминах l, t, r, b, в которой
416 размещается данный сегмент линии *)
417     BEGIN
418         IF x0 > x1 THEN l := x1; r := x0 ELSE l := x0; r := x1 END;
419         IF y0 > y1 THEN t := y1; b := y0 ELSE t := y0; b := y1 END;
420         INC( r, Ports.point ); INC( b, Ports.point )
421     END GetBox;
422
423     PROCEDURE ( v: View ) HandleCtrlMsg (* уже было для нажатий кнопок
424 клави *)
425     ( f: Views.Frame; VAR msg: Controllers.Message; VAR focus: Views.View );
426
427     VAR x0, y0, x1, y1, x, y, res, l, t, r, b: INTEGER;
428     modifiers: SET; isDown: BOOLEAN;
429     BEGIN
430         WITH msg: Controllers.EditMsg DO
431             (* реакция на клави: стандартная схема;
432 в данном случае реакция = меняем цвет *)
433             IF msg.op = Controllers.pasteChar THEN
434                 CASE msg.char OF
435                     | "B": color := Ports.black
436                     | "r": color := Ports.red
437                     | "g": color := Ports.green
438                     | "b": color := Ports.blue
439                 ELSE
440                     END
441             END
442             | msg: Controllers.TrackMsg DO (* посылается в момент нажатия кнопки
443 мыши *)
444                 (* положение, в котором была нажата кнопка мыши: *)
445                 x0 := msg.x; y0 := msg.y;
446
447                 (* дальше отслеживаем положение мышки пока нажата кнопка *)
448                 x1 := x0; y1 := y0; (* x1, y1 = старая позиция мыша *)
449                 (* у f есть спец. буфер, чтобы сложную картинку не перерисовывать.
450 Сохраним видимую часть картинки целиком в этот буфер: *)
451                 f.SaveRect( f.l, f.t, f.r, f.b, res ); (* операция успешна, если res = 0 *)
452                 IF res = 0 THEN
453                     f.DrawLine( x0, y0, x1, y1, Ports.point, color )
454                 END;
455             REPEAT
456                 (* x1, y1 = старая позиция мыша *)
457                 f.Input( x, y, modifiers, isDown ); (* "poll the mouse" = опросить *)
458                 (* isDown = все еще нажата;
459 x, y = где она в данный момент;
460 modifier = что еще нажато (Ctrl и т.д. *)
461                 IF ( x # x1 ) OR ( y # y1 ) THEN
462                     (* восстановим картинку, "испорченную" на пред. шаге: *)
463                     f.RestoreRect( f.l, f.t, f.r, f.b, Ports.keepBuffer );
464                     x1 := x; y1 := y;
465                     (* нарисуем линию в новом месте: *)
466                     IF res = 0 THEN
467                         f.DrawLine( x0, y0, x1, y1, Ports.point, color )
468                     END
469                 UNTIL ~isDown;
470                 (* восстановили картинку и "сбросили" буфер: *)
471                 f.RestoreRect( f.l, f.t, f.r, f.b, Ports.disposeBuffer );
472
473                 (* добавляем линию в список: *)
474                 v.Insert( x0, y0, x1, y1 );

```

```

473      (* на картинке в данный момент нет никакой новой линии -- линия
474      была видна, пока была нажата мышка; окончательно новая линия прорисовуется
475      при вызове Restore. Поскольку само по себе это вызовется только если что-то с
476      картинкой произойдет, то подскажем ББ: *)
477      Views.Update( v, Views.keepFrames );
478      ELSE
479      END
480  END HandleCtrlMsg;
481
482  PROCEDURE ( v: View ) HandlePropMsg ( VAR msg: Properties.Message );
483  BEGIN
484      WITH msg: Properties.FocusPref DO
485          msg.setFocus := TRUE (* вьюшка желает становиться фокусом *)
486      ELSE
487      END
488  END HandlePropMsg;
489
490  PROCEDURE Open*;
491  VAR v: View;
492  BEGIN NEW( v ); Views.OpenView( v )
493  END Open;
494
495  BEGIN color := Ports.black
496  END Kurs2006Mouse. ①Kurs2006Mouse.Open
497
498  Откроется окно, можно мышкой чертить отрезки прямых.
499  Нажимая r, g, b, B, меняем цвет будущих линий.
500
501  Упр Усовершенствовать так, чтобы можно было копировать и сохранять такие
502  вьюшки.
503
504  ObxLines: похожий, но более сложный пример. Там показывается, как
505  реализовать механизм Undo -- отмены последнего действия (удаления
506  последней нарисованной линии). Как только этот механизм задействован,
507  отменить можно любое количество действий.
508
509  Вьюшки внутри других вьюшек
510
511  В Блэкбоксе есть очень мощный механизм "контейнеров" (модуль Containers),
512  на котором основаны ББ-тексты и диалоговые формы. Там же, например,
513  поддержка механизма Undo. Не будем в такой полноте смотреть (нужен еще
514  семестр).
515
516  Посмотрим простейший способ вставить вьюшки в другие вьюшки.
517  Зачем? Чтобы иметь, скажем, FunctionViews, которые бы рисовали заданную
518  функцию кривыми Безье, а вещами типа полей (в смысле белое пространство у
519  картинки), рисование осей координат, наложение нескольких кривых друг на
520  друга и т.п. управлялись бы из независимой вьюшки-контейнера.
521
522  Сделаем так:
523  одна внешняя вьюшка, рисует рамочку.
524  Внутри себя, содержит две других вьюшки-близнецы, одинакового размера:
525  одна рисует одну диагональ в своем прямоугольнике, красным цветом, другая --
526  другую диагональ, зеленым цветом.

```

```

522  Размер и положение этих двух вьюшек задает внешняя вьюшка.
523
524  Для краткости модуля опустим HandlePropMsg для внешней вьюшки (т.е. она
525  никак не общается с ББ насчет своего размера).
526
527  MODULE Kurs2006Container;
528  IMPORT Log := StdLog, Math, In := FVTsysIn, Models, Views, Ports, Stores;
529
530  TYPE
531      (* тип для внутренних вьюшек (могло бы быть и несколько разных): *)
532      View = POINTER TO RECORD ( Views.View )
533          n: INTEGER; (* вьюшки отличаются этим значением *)
534      END;
535
536      (* спец. объект для передачи информации во внутренние вьюшки: *)
537      Context = POINTER TO RECORD ( Models.Context )
538          hsize, vsize: INTEGER; (* здесь будем хранить информацию о размере
539      внутр. вьюшки *)
540          v: View (* это и есть внутр. вьюшка -- где-то надо хранить на нее
541      ссылку, почему бы не здесь *)
542      END;
543
544      (* тип для внешней вьюшки: *)
545      Container = POINTER TO RECORD ( Views.View )
546          size, (* квази-размер самой внешней вьюшки *)
547          offx, offy: INTEGER; (* положение левого верхнего угла внутренних
548      вьюшек во фрейме внешней *)
549          c0, c1: Context
550      END;
551
552  PROCEDURE ( v: View ) Restore ( f: Views.Frame; l, t, r, b: INTEGER );
553      (* Как рисуется внутр. вьюшка. *)
554      VAR w, h : INTEGER; c: Ports.Color;
555      BEGIN
556          (* Сначала тандартное обращение к "контексту", чтобы узнать свой
557      размер -- чтобы, например, ужать свою картинку соответственно.
558      ББ точно также узнает про размер этой вьюшки -- ведь поле context
559      предопределено в ББ.
560      ББ должен это знать, чтобы предоставить вьюшке f: Views.Frame --
561      по-грубому, кусок видео памяти или памяти принтера, в котором вьюшка себя
562      будет изображать. Именно благодаря этому ББ не даст вьюшке нарисоваться за
563      пределами выделенного участка экрана.
564      *)
565          v.context.GetSize( w, h );
566          (* теперь вьюшка себя рисует в зависимости от содержимого: *)
567          CASE v.n OF
568              | 0 :
569                  f.DrawLine( 0, 0, w, h, 0, Ports.green )
570              | 1 :
571                  f.DrawLine( 0, h, w, 0, 0, Ports.red )
572          END;
573      END Restore;
574
575

```

```

570 (* Оформим "протокол" для Context *)
571
572 PROCEDURE ( c: Context ) GetSize ( OUT w, h: INTEGER );
573 BEGIN w := c.hsize; h := c.vsize (* главное, что мы знаем, откуда брать w и
574 h -- сами так решили. Могли бы решить и по-другому: иметь здесь ссылку на
575 контейнер, а там какой-нить список с размерами для внутренних вьюшек и т.п.
576 -- как угодно, лишь бы GetSize выдавала нужную инфу. *)
577 END GetSize;
578
579 PROCEDURE ( c: Context ) Normalize (): BOOLEAN;
580 (* Эта штука связана с метками вроде курсора, отмеченных кусков текста и
581 т.п. Заполним как-нибудь. Здесь не важно. *)
582 BEGIN RETURN TRUE
583 END Normalize;
584
585 PROCEDURE ( c: Context ) ThisModel (): Models.Model;
586 (* Вьюшка может иметь данные в отдельном объекте-модели.
587 Данный метод позволяет внутренней вьюшке узнать, во что она
588 погружена, чтобы по-разному реагировать. В данном случае "container does not
589 disclose its identity to the embedded view" = контейнер отказывается говорить
590 внутренней вьюшке о себе -- тем более, что и модели-то отдельной никакой нет.
591 Поскольку мы здесь сами контролируем, как ведет себя внутр. вьюшка,
592 мы вообще не будем вызывать этот метод.
593 Но ББ для единообразия требует, чтобы метод был. *)
594 BEGIN RETURN NIL
595 END ThisModel;
596
597 PROCEDURE ( v: Container ) Restore ( f: Views.Frame; l, t, r, b: INTEGER );
598 (* Рисование внешней вьюшки. Еще раз: Views.Frame -- это объект, через
599 который вьюшка себя рисует; сей объект может скрывать в себе принтер или
600 видеопамять или еще что-нить *)
601 BEGIN
602 (* сначала рисуем квадратную рамочку -- задача внешней вьюшки: *)
603 f.DrawRect( 0, 0, v.size, v.size, 0, Ports.black );
604 (* могли бы быть какие-нить координатные оси. *)
605
606 (* потом создаем два фрейма -- для каждой из внутр. вьюшек: *)
607 Views.InstallFrame( f, v.c0.v, v.offx, v.offy, 0, TRUE (* focus *) );
608 (* f -- что новый фрейм будет внутри f -- ББ поддерживает иерархию
609 фреймов.
610 v.c0.v -- какая вьюшка будет рисоваться через этот новый фрейм.
611 v.off*-- координаты лево-верхнего угла внутр. вьюшки в координатах
612 внешнего фрейма. Раз отступ, то будут поля -- в отличие от примера в прошлой
613 лекции, внутр. вьюшка не сможет нарисоваться за границами своего фрейма.
614 focus -- сообщает ББ, что внутр. вьюшка может стать фокусом и
615 принимать сигналы от клави и мыша, но мы этим не будем здесь для краткости
616 пользоваться.
617 *)
618
619 (* Нигде нет про размер фрейма внутренней вьюшки! Но и не надо:
620 эта информация передается через Context, о котором ББ все знает.
621 Данная процедура уже предполагает, что для внутр. вьюшек контекст
622 "установлен".
623 *)

```

```

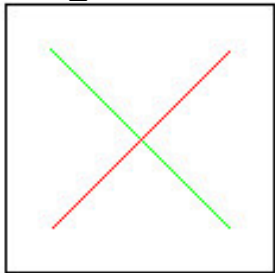
623 (* И вторая внутр. вьюшка: *)
624 Views.InstallFrame( f, v.c1.v, v.offx, v.offy, 0, TRUE (* focus *) )
625 (* А могли бы и в другом месте ее вставить, например, бок-о-бок с
626 первой -- мы же всё про все вьюшки знаем. *)
627 END Restore;
628
629 (* И, наконец, связываем все вместе: *)
630 PROCEDURE Do*;
631 VAR v: Container; v0, v1: View; c0, c1: Context;
632 BEGIN
633 (* просто создаем внешнюю вьюшку и задаем ее параметры: *)
634 NEW( v );
635 v.size := 30 * Ports.mm;
636 v.offx := 5 * Ports.mm;
637 v.offy := 5 * Ports.mm;
638
639 (* создаем первую из внутр. вьюшек: *)
640 NEW( v0 );
641 v0.n := 0;
642 Stores.Join( v, v0 ); (* Обязательный ритуал.
643 Все, что рисуется в одном окне, должно быть "повязано" друг с другом
644 посредством этого Join.
645 Называется "присоединить к домену". Достаточно сделать Join новой
646 вьюшки к любой из тех, что уже были присоединены -- не надо ко всем!!!
647 В общем случае: домен = Stores.Domain является представителем
648 документа, видимого в окне. Какой-то внутр. список ББ. Именно его ББ
649 использует, чтобы передавать информацию из внутренних вьюшек "наружу" --
650 ведь внутренние могут и не знать, где и на каком уровне глубины вложенности
651 они суть embedded.
652 Передавать инфу наружу может понадобится, когда вьюшка решит
653 изменить свой размер, например, в ответ на нажатие кнопки в диалоге и т.п.
654 Для передачи инфы наружу нужно вызвать процедуру
655 Models.Domaincast( domain, VAR message ), которая шлет заданный мессидж
656 всем вьюшкам домена -- а дальше наше дело, как внешняя вьюшка -- как и
657 вторая внутренняя -- будут реагировать на этот мессидж (HandleModelMsg). И
658 наше же дело -- определить этот мессидж. Например, правая вьюшка может
659 расширяться, а левая -- ужаться так, чтобы суммарная ширина не изменилась.
660 Причем управлять степенью ужатия может внешняя вьюшка -- например, чтобы
661 не стать шире экрана.
662 ББ открывает внешнюю вьюшку в окне -- и создает домен, и записывает
663 ссылку во внутр. поле вьюшки. Join обеспечивает, что все повязанные вьюшки
664 будут иметь один домен.
665 Кстати, доступ к домену -- v.Domain().
666 В нашем простом случае мы все связи и так знаем. Но это простой
667 случай...
668 *)

```

```

668      (* Пвяжем внешнюю вьюшку через контекст с внутренней: *)
669      NEW( c0 );
670      c0.hsize := 20 * Ports.mm;
671      c0.vsize := 20 * Ports.mm;
672      v0.InitContext( c0 ); (* теперь v0.context = c0 *)
673      c0.v := v0;
674
675      (* Наконец, "вставим" контекст во внешнюю вьюшку: *)
676      v.c0 := c0;
677
678      (* Сделаем аналогичные вещи для второй внутр. вьюшки: *)
679      NEW( v1 );
680      v1.n := 1;
681      Stores.Join( v, v1 );
682      NEW( c1 );
683      c1.hsize := 20 * Ports.mm;
684      c1.vsize := 20 * Ports.mm;
685      v1.InitContext( c1 );
686      c1.v := v1;
687      v.c1 := c1;
688
689      (* Откроем в окне: *)
690      Views.OpenView( v )
691  END Do;
692  END !Kurs2006Container.Do

```



```

693
694

```