

С/к Введение в современное программирование (v.5.5) Физфак МГУ. 2006/7 уч. год. Лекция 12

(Quasi) Напоминание

```

7 MODULE Kurs2006Ex39;
8   IMPORT Log := StdLog, Math, In := FVTsysIn;
9   TYPE
10     Directory* = POINTER TO ABSTRACT RECORD END;
11     StdDirectory = POINTER TO RECORD ( Directory ) END;
12     Complex* = POINTER TO ABSTRACT RECORD END;
13     StdComplex = POINTER TO RECORD ( Complex )
14       x, y: REAL
15     END;
16   VAR dir-, stdDir-: Directory;
17   PROCEDURE ( d: Directory ) New* ( x, y: REAL ): Complex, NEW, ABSTRACT;
18   PROCEDURE ( d: StdDirectory ) New ( x, y: REAL ): Complex;
19     VAR res: StdComplex;
20     BEGIN ASSERT( ( ABS( x ) # INF ) & ( ABS( y ) # INF ), 20 );
21       NEW( res ); res.x := x; res.y := y; RETURN res
22     END New;
23   PROCEDURE ( a: Complex ) Plus* ( b: Complex ): Complex, NEW, ABSTRACT;
24   PROCEDURE ( a: Complex ) GetXY* ( OUT x, y: REAL ), NEW, ABSTRACT;
25   PROCEDURE ( a: StdComplex ) Plus ( b: Complex ): Complex;
26     VAR res: StdComplex;
27     BEGIN
28       WITH b: StdComplex DO
29         NEW( res ); res.x := a.x + b.x; res.y := a.y + b.y;
30       END;
31     RETURN res
32   END Plus;
33   PROCEDURE ( a: StdComplex ) GetXY ( OUT x, y: REAL );
34     BEGIN x := a.x; y := a.y
35   END GetXY;
36   PROCEDURE SetDir* ( d: Directory );
37     BEGIN ASSERT( d # NIL, 20 ); dir := d
38   END SetDir;
39   PROCEDURE Init;
40     VAR d: StdDirectory;
41     BEGIN NEW( d ); stdDir := d; dir := d
42   END Init;
43   BEGIN Init
44   END Kurs2006Ex39.

```

45 **NB** Это всего лишь **демонстрация** механизмов -- **не** предложение делать
46 комплексную арифметику именно так.

```

47
48 DEFINITION Kurs2006Ex39;
49   TYPE
50     Complex = POINTER TO ABSTRACT RECORD
51       (a: Complex) Plus (b: Complex): Complex, NEW, ABSTRACT
52     END;
53
54     Directory = POINTER TO ABSTRACT RECORD
55       (d: Directory) New (x, y: REAL): Complex, NEW, ABSTRACT
56     END;
57
58   VAR
59     dir-: Directory;
60     stdDir-: Directory;
61
62   PROCEDURE SetDir (d: Directory);
63
64   END Kurs2006Ex39.

```

Комментарии

63 **--0--** Тип Complex -- интерфейсный.
64 Конкретная реализация -- StdComplex.
65 Понятие (еще один "паттерн"):

Разделение интерфейса и реализации

66 Separation of interface and implementation

68 **NB** Прикладная программа (клиент) пишется так, что она не знает ничего,
69 кроме заявленного интерфейса.
70 **NB** Возможность независимо улучшать реализацию, не "ломаю" клиентов.

71 **--1--** dir -- "**фабричный объект**", несущий "фабричную функцию" New.
72 Введен для того, чтобы "на лету" менять реализацию, не перегружая систему.
73 Для этого его интерфейс тоже абстрактен -- тип Directory (название --
74 историческое).
75 Для подмены фабричного "на лету" объекта дана спец. процедура.

76 **--2--** Всегда можно дать другую реализацию в другом модуле вот с таким
77 интерфейсом:

```

78 DEFINITION Kurs2006Ex39Another;
79   PROCEDURE Config*
80   END Kurs2006ExAnother.

```

81 **Q** Что делает Config? Почему ничего больше тут не видно?

82 **NB** Можно и Std определить в другой модуль (пример: Files, HostFiles).

83 **Q** Как тогда установить stdDir?

84 **NB** Наличие stdDir позволяет играть с новыми реализациями, продолжая
85 пользоваться старой, проверенной (пример: разработка текстов: старая
86 реализация -- просто тексты; новая реализация -- тексты с форматированием;
87 со вставными объектами вроде командеров; очень большие тексты...).

88

```

89 --3-- Предполагается, что клиенты пользуются только одним dir: этот dir --
90 "одиночка"/"singleton" ("паттерн Singleton").
91
92 пауза -- интернализировать.....
93 .....
94 Снова развилка: райдеры или HandleMsg...
95 Сейчас кратчайшим путем к Files.

```

96 Разделение абстракций хранения и доступа 97 Carrier/Rider

```

98 Вернемся к спискам.
99 TYPE
100   Link = POINTER TO ABSTRACT RECORD
101   next-: Link
102   END;

103 Видимость next экспонирует "кухню". Плохо.
104 Что "плохо"? Плохо масштабируется (напр., как хранить данные на диске).
105 Можно не экспортировать -- но как тогда "ходить" по данным?

106 NB тут мы уже говорим не о "списке", а об организации "набора данных" -- а
107 список лишь конкретная техника реализации.

108 Например, ходить могли бы так:
109   PROCEDURE ( l: List ) Next (): Link;  (* NIL = сигнал конца *)

110 Но тогда List должен таскать в себе информацию до текущего состоянии "хода"
111 по списку.
112 Если набор данных стал большим, то доступ к нему -- целая морока (открытые
113 файлы и т.п.). Все это таскать с собой?

114 Нужно обеспечить "доступ". Понятие <--> тип.

115 TYPE
116   Datum = POINTER TO ... (* элемент данных *)
117   Data = POINTER TO ... (* набор данных -- абстракция "хранения" -- carrier *)
118   Reader = POINTER TO ... (* "читатель" -- абстракция "доступа" -- rider *)

119   PROCEDURE ( d: Data ) NewReader (): Reader;

120   PROCEDURE ( rd: Reader ) Next (): Datum;

121 NB Приятность: программа, написанная в терминах абстрактного Reader'a,
122 может работать с конкретным Reader'ом, не имеющим соотв. carrier'a.
123 (Например, для тестов.)

124 Сильно упрощенный пример для иллюстрации идеи.
125 Этапы существования набора данных:

126 0) несуществование
127 1) последовательное создание
128 2) существование
129 3) последовательное чтение

130 dir --> Writer --> (+Datum)^n --> Data --> Reader
131

```

```

132 MODULE Kurs2006Ex40;
133 IMPORT Log := StdLog, In := FVTsysIn, Math;
134 TYPE
135   Datum* = POINTER TO ABSTRACT RECORD END;
136   Directory* = POINTER TO ABSTRACT RECORD END;
137   Data* = POINTER TO ABSTRACT RECORD END;
138   Writer* = POINTER TO ABSTRACT RECORD END;
139   Reader* = POINTER TO ABSTRACT RECORD END;

140   StdDirectory = POINTER TO RECORD ( Directory ) END;
141   Link = POINTER TO RECORD
142     next: Link;
143     d: Datum;
144   END;
145   StdData = POINTER TO RECORD ( Data )
146     first: Link;
147     len: LONGINT;
148   END;
149   StdWriter = POINTER TO RECORD ( Writer )
150     first, last: Link;
151     len: LONGINT;
152     open: BOOLEAN;
153   END;
154   StdReader = POINTER TO RECORD ( Reader )
155     next: Link;
156     finished: BOOLEAN;
157   END;

158   PROCEDURE ( d: Directory ) NewWriter* (): Writer, NEW, ABSTRACT;
159   PROCEDURE ( wr: Writer ) Write* ( d: Datum ), NEW, ABSTRACT;
160   PROCEDURE ( wr: Writer ) Close* ( OUT data: Data ), NEW, ABSTRACT;
161   PROCEDURE ( d: Data ) Len* (): LONGINT, NEW, ABSTRACT;
162   PROCEDURE ( d: Data ) NewReader* (): Reader, NEW, ABSTRACT;
163   PROCEDURE ( rd: Reader ) Next* (): Datum, NEW, ABSTRACT;

164   PROCEDURE ( d: StdDirectory ) NewWriter (): Writer;
165   VAR res: StdWriter;
166   BEGIN
167     NEW( res );
168     res.first := NIL; res.last := NIL; res.len := 0; res.open := TRUE;
169     RETURN res
170   END NewWriter;

171   PROCEDURE ( wr: StdWriter ) Write ( d: Datum );
172   VAR l: Link;
173   BEGIN ASSERT( wr.open, 20 ); ASSERT( d # NIL, 21 );
174     NEW( l ); l.d := d;
175     IF wr.first = NIL THEN
176       wr.first := l; wr.last := l;
177     ELSE
178       wr.last.next := l; wr.last := l
179     END;
180     INC( wr.len )
181   END Write;

```

```

182 PROCEDURE ( wr: StdWriter ) Close ( OUT data: Data );
183   VAR res: StdData;
184 BEGIN
185   NEW( res ); res.first := wr.first; res.len := wr.len;
186   wr.first := NIL; wr.last := NIL; wr.len := 0; wr.open := FALSE;
187 END Close;

188 PROCEDURE ( d: StdData ) Len (): LONGINT;
189 BEGIN RETURN d.len
190 END Len;

191 PROCEDURE ( d: StdData ) NewReader (): Reader;
192   VAR res: StdReader;
193 BEGIN
194   NEW( res ); res.next := d.first; res.finished := FALSE;
195   RETURN res
196 END NewReader;

197 PROCEDURE ( rd: StdReader ) Next (): Datum;
198   VAR res: Datum;
199 BEGIN ASSERT( ~rd.finished, 20 );
200   IF rd.next = NIL THEN
201     res := NIL; rd.finished := TRUE;
202   ELSE
203     res := rd.next.d;
204     rd.next := rd.next.next
205   END;
206   RETURN res
207 END Next;

208 END Kurs2006Ex40.
209
210 DEFINITION Kurs2006Ex40;

211 TYPE
212   Data = POINTER TO ABSTRACT RECORD
213     (d: Data) Len (): INTEGER, NEW, ABSTRACT;
214     (d: Data) NewReader (): Reader, NEW, ABSTRACT
215   END;

216   Datum = POINTER TO ABSTRACT RECORD END;

217   Directory = POINTER TO ABSTRACT RECORD
218     (d: Directory) NewWriter (): Writer, NEW, ABSTRACT
219   END;

220   Reader = POINTER TO ABSTRACT RECORD
221     (rd: Reader) Next (): Datum, NEW, ABSTRACT
222   END;

223   Writer = POINTER TO ABSTRACT RECORD
224     (wr: Writer) Close (OUT data: Data), NEW, ABSTRACT;
225     (wr: Writer) Write (d: Datum), NEW, ABSTRACT
226   END;

227 END Kurs2006Ex40.
228

```

229 — Не может быть двух Writer'ов на один набор данных.
 230 — Может быть несколько ридеров -- в это время набор данных
 231 модифицироваться не может.

232
 233 **NB** Разные фазы жизни набора данных отражены **статически**, т.е. через
 234 разные типы: невозможно выполнить неправильную операцию в не той фазе --
 235 компилятор не даст!
 236 **NB** Такое разделение не всегда возможно, но к нему надо стремиться в
 237 соответствии с принципом:
 238

239 **Максимум информации о задаче и алгоритмах**
 240 **должен быть выражен статически.**

241
 242 **NB** Этот лишь пример. Много деталей и вариаций:
 243 — нужны средства сохранения на диск и т.п.
 244 — Datum = ... RECORD next: Datum END; ([гораздо] меньше мусора, но доп.
 245 проблемы: ASSERT(d.next = NIL, 30); и т.п.).
 246 **NB** Типичное противоречие: простота (и надежность) <--> нагрузка на сборщик
 247 мусора.
 248 Это конкретное решение отражается на алгоритмах клиентов (можно ли один
 249 Datum включать в разные наборы данных).

250
 251 **NB** Если мы решили минимизировать сбор мусора, то можно повторное
 252 использование:

253
 254 (d: Data) NewReader (old: Reader): Reader;
 255 0) заявить в интерфейсе, но не реализовывать;
 256 1) реализовать *потом* (откладывание оптимизаций).
 257 Преждевременная оптимизация -- большой источник проблем. (Усложнение
 258 еще не устоявшегося кода, порождает к трудностям отладки при модификации.)
 259

260 **NB** Все время речь о принципе **DIVIDE ET IMPERE**.

261 **Q** В приведенном примере перечислить все приемы реализации этого принципа.
 262

263 **Работа с файлами и ввод-вывод (I/O).**

264 **Модуль Files**

265 Модуль Files — это уже "библиотека **типов** и процедур". Причем с вывертом
 266 (реализовано т.наз. "разделение интерфейса и реализации").
 267 Отличия от примера с Data
 268 -- вместо абстрактных Datum'ов -- конкретные байты, но зато
 269 последовательности.
 270 -- у традиционных файлов разрешается писать в любое место, а также читать и
 271 писать одновременно.

272
 273 Интерфейс устрашает с непривычки, быстро посмотрим и будем смотреть
 274 примеры, чтобы понять организацию.
 275 Рассматриваем только **подмножество** средств.
 276

277 Файл — конечная последовательность байтов. ОС предлагает какую-нить
 278 систему хранения файлов (средства управления и т.п.) — папки и т.п. Обычно
 279 файлы и папки идентифицируются по именам.
 280 Средства модуля Files -- только побайтовый ввод/вывод.
 281 Ввод/вывод INTEGER и т.п. -- следующий слой.
 282 **NB** Хороший пример расслоения -- еще один пример реализации принципа
 283 DIVIDE ET IMPERA.

284 Объект типа Files.File — представляет в памяти файл.
 285 Объект типа Files.Locator — представляет папку, в которой могут быть файлы
 286 или другие папки.

287 Обзор интерфейса модуля:

288 DEFINITION Files;

289 CONST

290 archive = 3; --атрибуты файлов

291 ask = TRUE;

292 dontAsk = FALSE;

293 exclusive = FALSE;

294 hidden = 1;

295 readOnly = 0;

296 shared = TRUE;

297 ...

298 TYPE

299 Directory = POINTER TO ABSTRACT RECORD

300 (d: Directory) **Delete** (loc: Locator; name: Name), ...;

301 (d: Directory) **FileList** (loc: Locator): FileInfo, ...; --список файлов

302 (d: Directory) **GetFileName** (name: Name; type: Type; OUT filename:

303 Name), ...;

304 (d: Directory) **LocList** (loc: Locator): LocInfo, ...; --список подпапок

305 (d: Directory) **New** (loc: Locator; ask: BOOLEAN): File, ...;

306 (d: Directory) **Old** (loc: Locator; name: Name; shared: BOOLEAN): File, ...;

307 (d: Directory) **Rename** (loc: Locator; old, new: Name; ask:

308 BOOLEAN), ...;

309 ...

310 (d: Directory) **Temp** (): File, ...;

311 (d: Directory) **This** (IN path: ARRAY OF CHAR): Locator, ...

312 END;

313

314 File = POINTER TO ABSTRACT RECORD

315 type-: Type;

316 (f: File) **Close**, ...; --если больше не нужен

317 ...

318 (f: File) **InitType** (type: Type), NEW; --конкретный метод!

319 (f: File) **Length** (): INTEGER, ...;

320 (f: File) **NewReader** (old: Reader): Reader, ...; (* ставьте NIL для

321 old *)

322 (f: File) **NewWriter** (old: Writer): Writer, ...;

323 (f: File) **Register** (name: Name; type: Type; ask: BOOLEAN; OUT res:

324 INTEGER), ...

325 END;

326 FileInfo = POINTER TO RECORD --обычный список!

327 **next**: FileInfo;

328 name: Name;

329 length: INTEGER;

330 type: Type;

331 modified: RECORD

332 year, month, day, hour, minute, second: INTEGER

333 END;

334 attr: SET

335 END;

336 LocInfo = POINTER TO RECORD --обычный список!

337 **next**: LocInfo;

338 name: Name;

339 attr: SET

340 END;

341 Locator = POINTER TO ABSTRACT RECORD

342 res: INTEGER;

343 (l: Locator) **This** (IN path: ARRAY OF CHAR): Locator, ... подпапка

344 END;

345 Reader = POINTER TO ABSTRACT RECORD

346 **eof**: BOOLEAN;

347 (r: Reader) **Base** (): File, ...;

348 (r: Reader) **Pos** (): INTEGER, ...;

349 (r: Reader) **ReadByte** (OUT x: BYTE), ...;

350 (r: Reader) **ReadBytes** (VAR x: ARRAY OF BYTE; beg, len:

351 INTEGER), ...;

352 (r: Reader) **SetPos** (pos: INTEGER), ...

353 END;

354 Writer = POINTER TO ABSTRACT RECORD

355 (w: Writer) **Base** (): File, ...;

356 (w: Writer) **Pos** (): INTEGER, ...;

357 (w: Writer) **SetPos** (pos: INTEGER), ...;

358 (w: Writer) **WriteByte** (x: BYTE), ...;

359 (w: Writer) **WriteBytes** (IN x: ARRAY OF BYTE; beg, len:

360 INTEGER), ...

361 END;

362 Name = ARRAY 256 OF CHAR;

363 Type = ARRAY 16 OF CHAR;

364 VAR

365 **dir** -: Directory;

366 docType -: Type;

367 objType -: Type;

368 stdDir -: Directory;

369 symType -: Type;

370

371 PROCEDURE SetDir (d: Directory);

372 END Files.

373





374 **как создать/стереть и т.д. файл**

```

375 MODULE Kurs2006Files;
376   IMPORT Log := StdLog, Files;
377   VAR path: ARRAY 1000 OF CHAR;
378   PROCEDURE Create*;
379     VAR loc: Files.Locator; (* объект для доступа к конкретной папке *)
380     f: Files.File; (* указатель на объект для доступа к конкр. файлу *)
381     res: INTEGER;
382   BEGIN
383     loc := Files.dir.This( path );
384     f := Files.dir.New( loc, Files.dontAsk ); (* конкретный файл для послед.
385 "регистрации" *)
386 (* NB модуль обычно экспортирует константы с мнемоническими именами для
387 возможных значений какого-то параметра, в данном случае BOOLEAN *)
388     f.Register( "test.txt", "", Files.dontAsk, res );
389     (* Files.ask -- открывает диалог, предлагая как default то, что здесь
390 задано *)
391     Log.Int( loc.res ); Log.Ln; (* loc.res -- код ошибки; 0 = все в порядке *)
392
393     f := Files.dir.Temp(); (* конкретный файл для временного пользования *)
394     (* после выхода из процедуры loc и все f становятся "мусором" и в
395 какой-то момент утилизируются, при этом открытые файлы автоматически
396 закрываются и т.п. *)
397   END Create;
398
399   PROCEDURE Open*;
400     VAR loc: Files.Locator; f: Files.File; res: INTEGER;
401   BEGIN
402     loc := Files.dir.This( path );
403     f := Files.dir.Old( loc, "test.txt", Files.shared ); (* Files.exclusive *)
404     Log.Int( f.Length() ); Log.Ln
405   END Open;
406
407   PROCEDURE Rename*;
408     VAR loc: Files.Locator;
409   BEGIN
410     loc := Files.dir.This( path );
411     Files.dir.Rename( loc, "test.txt", "test.zip", Files.dontAsk );
412     Log.Int( loc.res ); Log.Ln;
413   END Rename;
414
415   PROCEDURE Delete*;
416     VAR loc: Files.Locator;
417   BEGIN
418     loc := Files.dir.This( path );
419     Files.dir.Delete( loc, "test.zip" ); (* NB не сразу стирается *)
420     Log.Int( loc.res ); Log.Ln;
421   END Delete;
422
423 BEGIN path := "c:\temp"
424 END Kurs2006Files.

```

422 Откройте папку, указ. в path, и следите!

423 Kurs2006Files.Create Kurs2006Files.Open Kurs2006Files.Rename
424 Kurs2006Files.Delete

425
426 **Упр** Переименовать в заданной папке все файлы PIC0***.jpg в *.jpg

427 **Упр** То же, но во всех подпапках тоже.

428 **Упр** Привинтить диалог (папка, подпапки?, какие расширения).

429

430 **Схема чтения/записи в файл**

431

432 Как работать с **содержимым** файлов?

433 Нужно уметь читать-писать байты в заданных позициях файлов.

434 Например, Read/Write(f: Files.File; b: BYTE; pos: INTEGER).

435 Но по ряду соображений нехорошо. В частности, байты в файл обычно

436 пишутся/читаются подряд. Хорошо бы освободить клиента от обязанности

437 помнить текущую позицию. Просто бы он записывал подряд свои данные, а

438 потом параллельной последовательностью считывал.

439 **Кто** должен помнить о позиции?

440 Получаем концепт/абстракцию "доступа" — нечто новое по ср. с концептом
441 "хранения".

442

443 Поэтому для чтения/записи вводятся объекты спец. типов: Reader/Writer.

444 Такая схема/pattern (разделенные абстракции хранения/доступа) имеет

445 название **carrier/rider** (изобретено в Обероне). Имеет ряд сильных

446 преимуществ (гибкость и др.). Вернемся чуть позже.

447

448 Далее. Мы обычно хотим читать/писать INTEGER, REAL и т.д., а не байты. Соотв.

449 преобразования в обе стороны — еще одна новая (и независимая) концепция

450 (Mapper). Поэтому в Files пытаемся сделать хорошо одно дело: чтение/запись

451 байтов.

452 Преобразования — в другом модуле (Stores), чуть позже.

453 В конце концов, предсказать все возможные способы использования файлов

454 невозможно. А побайтные операции предоставляют максимальную гибкость.

455

456 Итак, в Files только базовые средства ввода-вывода — побайтные (зато можно

457 переписывать содержимое файла побайтно):

```

458 MODULE Kurs2006Files1;
459   IMPORT Log := StdLog, Files;

```

```

460   PROCEDURE Copy ( from, to: Files.File );
461     VAR rd: Files.Reader; wr: Files.Writer; b: BYTE;
462   BEGIN
463     ASSERT( ( from # NIL ) & ( to # NIL ) & ( to.Length() = 0 ), 20 );
464     rd := from.NewReader( NIL ); ASSERT( rd.Pos() = 0 );
465     wr := to.NewWriter( NIL ); ASSERT( wr.Pos() = 0 );
466     rd.ReadByte( b );
467     WHILE ~rd.eof DO
468       wr.WriteByte( b );
469       rd.ReadByte( b )
470     END;
471   END Copy;

```

```

472 PROCEDURE Do*;
473     CONST oldname = "paper.pdf"; newname = "copy.pdf";
474     VAR loc: Files.Locator; old, new: Files.File; res: INTEGER;
475 BEGIN
476     loc := Files.dir.This("c:\temp");
477     old := Files.dir.Old( loc, oldname, Files.shared );
478     ASSERT( loc.res = 0 );
479     new := Files.dir.New( loc, Files.dontAsk );
480     ASSERT( loc.res = 0 );
481     Log.Int( old.Length() ); Log.Ln;
482     Copy( old, new );
483     Log.Int( new.Length() ); Log.Ln;
484     ASSERT( old.Length() = new.Length() );
485     new.Register( newname, "", Files.dontAsk, res ); ASSERT( res = 0 );
486 END Do;

```

487 END **!Kurs2006Files1**.Do

488 Можно записывать/считывать данные подряд — Reader|Writer делают за нас
 489 работу по запоминанию текущей позиции.
 490 Но можно с помощью процедур SetPos() переписать любой фрагмент файла в
 491 любое место другого файла. В общем случае SetPos — **затратная** операция.
 492 Выделение в отдельную процедуру (в отличие, скажем, от просто параметра)
 493 подчеркивает этот факт.
 494

495 **Схема (Pattern): Carrier-Rider = Носитель-** 496 **Бегунок**

497 Pattern = паттерн, схема = схема организации совместной работы одной или
 498 нескольких программных компонент. Обычно речь идет об объектах. Но понятие
 499 полезно и в более широком смысле (когда объекты "растворяются" в
 500 программе).
 501 Схемы построения кода (полный проход, схема поиска и т.п.).
 502 Схема "Генератор" (инициализация, переход к след. "объекту", условие
 503 завершения) -- и для фрагментов кода, и для отдельного модуля (In), и для
 504 объектов (пример с "близнецами" -- сразу три варианта).

505 Здесь имеем Files.File и пару Reader/Writer. При этом разделены:
 506 File = абстракция хранения (Carrier = носитель). Операции и свойства — только
 507 для набора байт как целого.
 508 Reader/Writer = абстракции доступа (Rider = бегунок)

509 картинка
 510 картинка
 511 картинка
 512 картинка

513 При этом:

514 Здесь: Files.Reader -- типичный "генератор":
 515 функция rd := f.NewReader() — фабричная функция, возвращает готовый к
 516 работе ("инициализированный") Reader — аналог In.Open;
 517 rd.ReadByte(x) — аналог In.Int(x);
 518 rd.eof — аналог ~In.Done.
 519 Files.Writer — очень похож, но без явного ограничения (как у генератора случ.
 520 чисел). Невяное есть — Files.Length(): INTEGER.

521 **Связь между Носителем и Бегунком**

522 В нашем случае с помощью "фабричного метода": Files.NewReader/NewWriter.
 523 Reader/Writer могут сообщить о своем "носителе" с помощью функции
 524 rd.wr.Base().
 525 Здесь реализована связь в обе стороны через функции.

526 **Side remark** Параметр old -- чтобы поменьше мусору генерить. Обычно сначала
 527 предусматривается в интерфейсе, но для простоты игнорируется в первой реализации.
 528 Потом — при наличии реальной необходимости, времени и сил — реализуется повторное
 529 использование/reuse.
 530 **NB** Редко нужно. Не нужно вводить такой параметр в своих дизайнах только потому, что
 531 вы о нем знаете... KISS!!
 532 В Files введен, т.к. слишком уж широкий спектр применения, предсказать невозможно ...

533 Связывать Носитель с Бегунком можно и по-другому (см. Stores).

534 **NB** Детали меняются, схема остается.

535 **О разделении интерфейса и реализации**

536 Все типы, экспортированные из Files — ABSTRACT.

537 Конкретная реализация скрыта.

538 Схема наследования:

539 Files.File*/ABSTRACT ---> видит клиент

540 ---> Files.StdFile, наследует Files.File, реализует все методы, не виден

541 клиенту

542 но именно экземпляры этого скрытого конкретного типа получает клиент.

543 *****Упр** (Чтобы оценить, почему все типы, которые мы видим, ABSTRACT.) Дать
 544 custom реализацию File и т.п., так чтобы запись шла в большой ARRAY OF BYTE.
 545 Автоматически сможем подсоединять к такому File'y Stores.Writer/Reader (см.
 546 ниже) и записывать туда произвольные типы!

547 *****Упр** Используя процедуру Files.SetDir подставить свою собств. dir, так
 548 чтобы Files.dir.Temp() создавал файл, реализованный как массив в памяти.

549 *****Упр** Подумать о реализации и доступа к файлам, на самом деле сидящим в
 550 других узлах сети.

551 *****Упр** Подумать о реализации файловой системы, в которой длина файлов
 552 была бы LONGINT.

553 **Stores: Более развитые средства ввода-вывода**

554 В т.ч. ввод-вывод произвольных структур данных (необходимая поддержка).

555 DEFINITION Stores;

556 ...

557 Reader = RECORD (* конкретный тип, на стеке *)

558 rider-: Files.Reader; (* композиция *)

559 ...

560 (VAR rd: Reader) **ConnectTo** (f: Files.File), NEW;

561 (* ConnectTo(NIL) *)

562 (VAR rd: Reader) Pos (): INTEGER, NEW;

563 (VAR rd: Reader) ReadBool (OUT x: BOOLEAN), NEW;

564 (VAR rd: Reader) ReadByte (OUT x: BYTE), NEW;

565 (VAR rd: Reader) ReadChar (OUT x: CHAR), NEW;

566 (VAR rd: Reader) ReadInt (OUT x: INTEGER), NEW;

567 (VAR rd: Reader) ReadLong (OUT x: LONGINT), NEW;

568 (VAR rd: Reader) ReadReal (OUT x: REAL), NEW;

```

569 ( VAR rd: Reader ) ReadSChar ( OUT x: SHORTCHAR ), NEW;
570 ( VAR rd: Reader ) ReadSInt ( OUT x: SHORTINT ), NEW;
571 ( VAR rd: Reader ) ReadSReal ( OUT x: SHORTREAL ), NEW;
572 ( VAR rd: Reader ) ReadSString ( OUT x: ARRAY OF SHORTCHAR ), NEW;
573 ( VAR rd: Reader ) ReadSet ( OUT x: SET ), NEW;
574 ( VAR rd: Reader ) ReadStore ( OUT x: Store ), NEW;
575 ( VAR rd: Reader ) ReadString ( OUT x: ARRAY OF CHAR ), NEW; --NB
576 ...
577 ( VAR rd: Reader ) SetPos ( pos: INTEGER ), NEW;
578 ...
579 END;

580 Writer = RECORD
581   (* аналогично: Read.. --> Write.. *)
582 END;

583 END Stores.

584 Похоже на In.
585 (Здесь) правильный порядок считывания нужно знать заранее!
586 Сколько байт записал, wr.WriteInt( 3 ), ровно столько же прочтет rd.ReadInt( x ).

587 NB При хранении больших массивов данных (терабайты 2^40, петабайты 2^50
588 -- напр., данных от ускорителя LHC/CERN) -- м.б. полезно, например, не
589 хранить 8 байт для REAL, если там значащих цифр только на SHORTREAL и т.п.

590 NB Stores.Reader/Writer делают свои преобразования в байты — и отдают их в
591 "rider" внутри себя, который и делает чтение/запись байтов.
592 NB глагол to forward (operation)

593 Насладимся расслоением на абстракции: уровень байтов — уровень базовых
594 типов.

595 По аналогичной схеме - работа с текстами (позже).

596

597 Чтение/запись сложных структур данных

598 там еще есть очень важный абстрактный тип:

599   Store = POINTER TO ABSTRACT RECORD
600     ( s: Store ) CopyFrom- ( source: Store ), NEW, EMPTY;
601     ( s: Store ) Externalize- ( VAR wr: Writer ), NEW, EXTENSIBLE;
602     ( s: Store ) Internalize- ( VAR rd: Reader ), NEW, EXTENSIBLE
603   END;
604
605 Это ОЧЕНЬ ВАЖНОЕ средство для поддержки работы с динамич. структур
606 данных, конкретно для записи в файл.
607 Аналогично сбору мусора: отследить вручную, что записали, а что нет, трудно.
608

```

```

609 MODULE Kurs2006StoresSave;
610 (* спасаем и восстанавливаем из файла кольцевой список;
611   Domain не используется *)
612
613 IMPORT Files, Stores, Services;
614
615 TYPE
616   Store = POINTER TO RECORD ( Stores.Store )
617     n: INTEGER;
618     link: Link; (* кольцо *)
619     s: Store (* еще ссылка, чтобы запутать Блэббокс ☺ *)
620   END;
621
622   Link = POINTER TO RECORD (* никакого отношения к Stores *)
623     n: INTEGER;
624     store: Store; (* кольцо *)
625     l: Link (* а эту ссылку Блэббокс не восстановит — чудес нет ☹ *)
626   END;
627
628 VAR кольцо: Store; f: Files.File;
629
630 PROCEDURE ( l: Link ) Write ( VAR wr: Stores.Writer ), NEW;
631 BEGIN
632   wr.WriteInt( l.n );
633   wr.WriteStore( l.store );
634   (* l.l.Write( wr ) --> дурная рекурсия --> stack overflow *)
635 END Write;
636
637 PROCEDURE ( l: Link ) Read ( VAR rd: Stores.Reader ), NEW;
638   VAR s0: Stores.Store;
639 BEGIN
640   rd.ReadInt( l.n );
641   rd.ReadStore( s0 ); l.store := s0( Store );
642   (* NEW( l.l ); l.l.Read( rd ) *)
643 END Read;
644
645
646 PROCEDURE ( s: Store ) Externalize- ( VAR wr: Stores.Writer );
647 BEGIN
648   wr.WriteInt( s.n );
649   s.link.Write( wr );
650   wr.WriteStore( s.s )
651 END Externalize;
652
653 PROCEDURE ( s: Store ) Internalize- ( VAR rd: Stores.Reader );
654   VAR s0: Stores.Store;
655 BEGIN
656   rd.ReadInt( s.n );
657   NEW( s.link ); s.link.Read( rd );
658   rd.ReadStore( s0 ); s.s := s0( Store )
659 END Internalize;
660

```

```

661 PROCEDURE Create* (): Store;
662     VAR s0, s2: Store; l1, l3: Link;
663 BEGIN
664     NEW( s0 );   s0.n := 0;
665     NEW( l1 );   l1.n := 1;
666     NEW( s2 );   s2.n := 2;
667     NEW( l3 );   l3.n := 3;
668     (* формируем кольцевую структуру: *)
669     s0.link := l1; l1.store := s2; s2.link := l3; l3.store := s0;
670     (* формируем ссылки дополнит. ссылки: *)
671     s0.s := s2; s2.s := s0; l1.l := l3; l3.l := l1;
672     RETURN s0
673 END Create;

674 PROCEDURE New*;
675 BEGIN
676     f := Files.dir.Temp(); кольцо := Create()
677 END New;

678 PROCEDURE Save*;
679     VAR wr: Stores.Writer;
680 BEGIN
681     wr.ConnectTo( f );
682     (* запись ВСЕЙ кольцевой структуры в файл: *)
683     wr.WriteStore( кольцо ); (* бесконечного цикла нет, т.к. между
684 WriteStore и Externalize Блэкбокс делает нечто, что позволяет ему всё
685 правильно отследить *)

686     кольцо := NIL; Services.Collect (* уничтожили ее следы в памяти *)
687 END Save;

688 PROCEDURE Restore*;
689     VAR rd: Stores.Reader; s0: Stores.Store;
690 BEGIN
691     rd.ConnectTo( f );
692     (* восстановление из файла: *)
693     rd.ReadStore( s0 ); кольцо := s0( Store )
694 END Restore;

695 END Kurs2006StoresSave.

696 ! Kurs2006StoresSave.New
697 Info --> Global Variables: s = кольц. структура с перек. ссылками
698 ! Kurs2006StoresSave.Save
699 Update в окошке Variables: s = NIL
700 ! Kurs2006StoresSave.Restore
701 Update в окошке Variables: s = снова кольц. структура, по др. адресу,
702 перекрестные ссылки правильно восстановились — но только у Store's.
703
704 Между wr.WriteStore и Store.Externalize выполняются доп. действия, которые и
705 обеспечивают корректность.
706 Ср. с ABSTRACT-списками, когда сортировали: там тоже клиент предоставляет
707 процедуру (сравнения), которой сервер и пользуется.
708 Логика та же, но здесь сервер реализует более сложное деяние.
709
710

```

"reflection services"

712 -- ввод/вывод объектов заранее неизвестных типов и т.п.
 713 модули Services, Meta

715 **Services:** часть про reflection services — только для записей, но зато на стеке
 716 (на стеке теги по-другому устроены):

717
 718 DEFINITION Services;

719
 720 PROCEDURE GetTypeName (IN rec: ANYREC; OUT type: ARRAY OF CHAR);

721 модуль.тип

722 PROCEDURE SameType (IN ra, rb: ANYREC): BOOLEAN; точное равенство

723

724 PROCEDURE Extends (IN type, base: ARRAY OF CHAR): BOOLEAN;

725 PROCEDURE IsExtensionOf (IN ra, rb: ANYREC): BOOLEAN;

726 PROCEDURE Is (IN rec: ANYREC; IN type: ARRAY OF CHAR): BOOLEAN;

727 (extension of)

728

729 PROCEDURE Level (IN type: ARRAY OF CHAR): INTEGER; у нового типа 0

730 PROCEDURE TypeLevel (IN rec: ANYREC): INTEGER;

731 ...

732 PROCEDURE AdrOf (IN rec: ANYREC): INTEGER; машинный адрес

733 PROCEDURE Collect; форсирует сбор мусора

734 ...

735

736 END Services.

737

738 Модуль **Meta:** можно много чего: исследовать (dynamically = at run time) модуль
 739 и выяснить, какие процедуры он экспортирует и т.п.

740 Вот пример использования для воссоздания объекта неизвестного типа из
 741 файла:

742

743 MODULE **Kurs2006Meta**;

744 IMPORT Log := StdLog, Meta, Files, Stores;

745

746 TYPE **Term*** = POINTER TO RECORD

747 a: INTEGER;

748 x: REAL;

749 s: ARRAY 100 OF CHAR

750 END;

751

752 VAR l: Files Locator;

753

754 PROCEDURE **WriteFullTypeName** (VAR wr: Stores.Writer; obj: **ANYPTR**);

755 VAR i: Meta.Item; mod, type: Meta.Name; len: INTEGER; path: ARRAY

756 512 OF CHAR;

757 BEGIN

758 Meta.GetItem(obj, i);

759 i.GetTypeName(mod, type);

760 path := mod + '.' + type;

761 Log.String("before"); Log.Ln;

762 Log.String(path); Log.Ln;

763 Log.String(path\$); Log.Ln;

```

764   Log.Char( path[LEN( path$ ) - 1] ); Log.Ln;
765   (* последний символ ^, поэтому нужно укоротить: *)
766   path[ LEN( path$ ) - 1 ] := 0X;
767   Log.String("after"); Log.Ln;
768   Log.String( path ); Log.Ln;
769   Log.String( path$ ); Log.Ln;
770   Log.Char( path[LEN( path$ ) - 1] ); Log.Ln;

771   Log.String( path ); Log.Ln;
772   wr.WriteString( path )
773   END WriteFullTypeName;

774   PROCEDURE Write* ( name: ARRAY OF CHAR ); (* storing object to a file
775   together with object type *)
776   VAR f: Files.File; wr: Stores.Writer; t: Term; res: INTEGER;
777   BEGIN
778     f := Files.dir.New( l, Files.dontAsk );
779     wr.ConnectTo( f );

780     NEW( t ); t.a := 2004; t.x := 3.3333333; t.s := "asymptotic operation";

781     WriteFullTypeName( wr, t );

782     wr.WriteInt( t.a );
783     wr.WriteReal( t.x );
784     wr.WriteString( t.s );

785     wr.ConnectTo( NIL );
786     f.Register( name$, "", Files.dontAsk, res ); ASSERT( res = 0 )
787     END Write;

788
789   PROCEDURE Read* ( name: ARRAY OF CHAR ); (* recreating an object stored
790   in a file without static knowledge of type *)
791   VAR f: Files.File; rd: Stores.Reader; path: ARRAY 512 OF CHAR; i:
792   Meta.Item; t: Term;

793   BEGIN
794     f := Files.dir.Old( l, name$, Files.shared );
795     rd.ConnectTo( f );
796     rd.ReadString( path );

797     Meta.LookupPath( path, i ); ASSERT( i.Valid() );
798
799     t := i.New() ( Term );

800
801     rd.ReadInt( t.a );
802     rd.ReadReal( t.x );
803     rd.ReadString( t.s );
804     HALT(0)
805     END Read;
806
807   BEGIN l := Files.dir.This("")
808   END Kurs2006Meta.
809
810   !"Kurs2006Meta.Write('test')" !"Kurs2006Meta.Read('test')"
811

```

```

812   "потрохи":
813   TRAP 0
814
815   <> Kurs2006Meta.Read [0000039FH] <>
816   .f Files.File [01093270H] <>
817   .i Meta.Item =>fields<=
818   .name ARRAY 256 OF CHAR "test"
819   .path ARRAY 512 OF CHAR "Kurs2006Meta.Term"
820   .rd Stores.Reader =>fields=>
821   .t Kurs2006Meta.Term [010D0DE0H] <>
822   <> StdInterpreter.CallProc [0000047AH] <>
823   .a BOOLEAN FALSE
824   .....
825   в окошке Variables после раскрытия .t:
826   Kurs2006Meta.Read:t^
827
828   [010D0DE0H]Kurs2006Meta.Term^ ==>
829   .aINTEGER 2004
830   .xREAL 3.3333333
831   .sARRAY 100 OF CHAR "asymptotic operation" ==>
832   ....
833
834   NB немножко неполна документация:

835   MODULE Kurs2006MetaTest;
836   IMPORT Log := StdLog, Meta, Files;

837   TYPE
838     Term = POINTER TO RECORD END;
839     Rec = RECORD END;
840     Ptr = POINTER TO Rec;

841   PROCEDURE Show ( obj: ANYPTR ); (* то же, что и раньше *)
842   VAR i: Meta.Item; mod, type: Meta.Name;
843   BEGIN
844     Meta.GetItem( obj, i );
845     i.GetTypeName( mod, type ); Log.String( mod + '.' + type ); Log.Ln;
846   END Show;

847
848   PROCEDURE Do*;
849   VAR t: Term; ptr: Ptr;
850   BEGIN
851     NEW( t ); Show( t ); (* печатает Kurs2006MetaTest.Term^ *)
852     NEW( ptr ); Show( ptr ); (* печатает Kurs2006MetaTest.Rec *)
853   END Do;
854
855   END Kurs2006MetaTest.
856
857   !Kurs2006MetaTest.Do
858
859   Вспомнить про "базовый тип" для "указательного типа".
860

```