КАЗАН ФЕДЕРАЛЬ УНИВЕРСИТЕТЫ
ХИСАПЛАУ МАТЕМАТИКАСЫ ҺӘМ МӘГЪЛҮМАТИ ТЕХНОЛОГИЯЛӘР ИНСТИТУТЫ
Мәгълүмати системалар кафедрасы

Ә.Ф. ГАЛИМҖАНОВ, Ч.Б. МИҢНЕГАЛИЕВА

ОБЪЕКТКА ОРИЕНТЛАШКАН ПРОГРАММАЛАУГА КЕРЕШ

Уку әсбабы

Казан – 2016

УДК 004.4

Хисаплау математикасы һәм мәгълүмати технологияләр институтының нәшрият советы карары,
протокол № 4, 2016 ел.

Мәгълүмати системалар кафедрасы утырышы карары,
протокол № 9, 25 май, 2016 ел
буенча басыла

Рецензентлар:

физика-математика фәннәре кандидаты,
мәгълүмати системалар кафедрасы доценты **Ф.М. Гафаров**,
педагогика фәннәре кандидаты, КМТТУ доценты
К.К. Исмәгыйлева.

Галимянов А.Ф., Миннегалиева Ч.Б. Введение в объектно-ориентированное программирование: учебное пособие / А.Ф. Галимянов, Ч.Б. Миннегалиева. – Казань: Казан.ун-т, 2016 . – 141 с.

Уку әсбабы югары уку йортлары студенталырына тәкъдим ителә. Әсбап объектка ориентлашкан программалауның фундаменталь нигезләре һәм мөһим принциплары белән таныштыра. Мисаллар Delphi мохитендә эшләнгән.

© Галимянов А.Ф., Миннегалиева Ч.Б., 2016
© Казанский университет, 2016

ЭЧТӘЛЕК

Кереш	5
1. Программ тәэминат төзүнең катлаулылыгы	6
1.1. Декомпозиция	6
1.2. Программалау телләренең кыскача тарихы	7
1.3. Программалауның төп парадигмаларына кыскача күзәтү	10
2. Delphi да программа структурасы	12
2.1. Delphi проекты	12
2.1.1. Проект төшенчәсе	12
2.1.2. Формаларны тасвирлау файллары	13
2.1.3. Программа модульләре файллары	17
2.1.4. Проектның төп файлы	22
2.1.5. Проектның башка файллары	23
2.2. Проект белән идарә итү	24
2.2.1. Проектны төзү, саклау һәм ачу	24
2.2.2. Проект белән идарә тәрәзе	26
2.2.3. Проектлар төркеме	29
2.2.4. Проектның параметрларын көйләү	30
2.2.5. Проектны компиляцияләү һәм җыю	37
2.2.6. Әзер кушымтаны эшләтеп җибәрү	38
2.2.7. Модуль структурасы	38
2.3. Динамик библиотекалар (тупламнар)	51
2.3.1. Динамик йөкләнүче библиотекалар (тупламнар)	51
2.3.2. Библиотеканы төзү	52
2.3.3. Библиотеканы программада куллану	60
2.3.4. С++ телендәге программадан библиотеканы куллану	67
2.3.5. Глобаль үзгәрешлеләр һәм константалар	68
2.3.6. Инициализация һәм библиотека эшен тәмамлау	68
2.3.7. Гадәттән тыш очраклар һәм астпрограмма башкару хаталары	70
2.3.8. Хәтернең уртак идарә программасы (менеджер)	70
3. Объектка ориентлашкан программалауның (ООП) төп принциплары	71
3.1. Класс билгеләмәсе	73
3.2. Инкапсуляция	75
3.3. Мирас итеп алу	75
3.4. Класслар диаграммасы	78

3.5. Классны игълан итү 81
3.6. Виртуаль ысуллар һәм полиморфизм 84
3.6.1. Статик һәм виртуаль ысуллар 84
3.6.2. Берничә конструкторлы класслар 89
3.7. Яңадан йөкләнә ала торган методлар 94
3.8. Ысулларны делегирлау 95
3.9. Үзлекләр 99
3.10. Үзлекләр массивлары 102
3.11. Индекс спецификаторлары 111
3.12. Класс ысуллары һәм класс күрсәткечләре 114
3.13. Класслар белән гамәлләр 117
4. Гадәттән тыш очракларны эшкәртү 122
4.1. Гадәттән тыш очрак төшенчәсе, аны Delphi чаралары белән эшкәртү 123
4.2. RTL-гадәттән тыш очракларын эшкәртү. Гадәттән тыш очракларның иерархиясе 128
4.3. Үз гадәттән тыш очракларыңны төзү 133
5. Интерфейслар һәм аларны класслар тарафыннан бергә куллану 134
5.1. IUnknown интерфейсы 135
5.2. TInterfacedObject классы 137
5.3. as операторын куллану 137
5.4. implements ачкыч сүзен куллану 138
5.5. Интерфейсларны бүленгән кушымталарда куллану 139
Әдәбият 141

КЕРЕШ

Объектка ориентлашкан программалау (ООП) – хәзерге заман программалау ысулы, аны кулланганда программаның төп элементлары булып объектлар тора. Бу ысулны кулланучы программалау телләре моннан кырык елдан элегрәк барлыкка килә, бүгенге көндә ООП күпчелек проектларда кулланыла.

Уку әсбабы объектка ориентлашкан программалауның фундаменталь нигезләре һәм мөһим принциплары белән таныштыра. ООПның төшенчәләрен һәм ысулларын аңлатучы мисаллар Delphi мохитендә Object Pascal программалау теле кулланып эшләнгән. Pascal теле урта мәктәптә программалау нигезләрен өйрәтә башлаганда еш кулланыла, Object Pascal – ул Pascal теленең объектлы киңәйтелүе. Шуңа Object Pascal нең югары уку йортларында укыту барышында кулланылуы нигезле. Бу телнең хәзер (бик корректлы булмаса да) Delphi теле дип аталуын әйтеп китик, әсбапта укыту күчүчәнлеген саклау максатыннан Object Pascal исеме калдырылды.

Delphi 1 эш мохите (Borland Delphi) 1995 елда Borland компаниясе тарафыннан эшләнелә. Аның соңгы версиясе булып Delphi 10 тора, ул 2015 елдан кулланучыларга таныш, Embarcadero Technologies компаниясе чыгарган. Бу әсбаптагы мисаллар Delphi 7 мохите өчен эшләнгән, чөнки ул, 2002 елда чыкканнан башлап, иң еш кулланылучы уңышлы продукт булып тора.

Хәзерге вакытта укыту процессында Lazarus программ тәэминат эшләү мохите дә киң кулланыла, ул Object Pascal гә нигезләнгән. Lazarusның формалар редакторы һәм объектлар инспекторы Delphi мохитенекенә охшаш.

1. ПРОГРАММ ТӘЭМИНАТ ТӨЗҮНЕҢ КАТЛАУЛЫЛЫГЫ

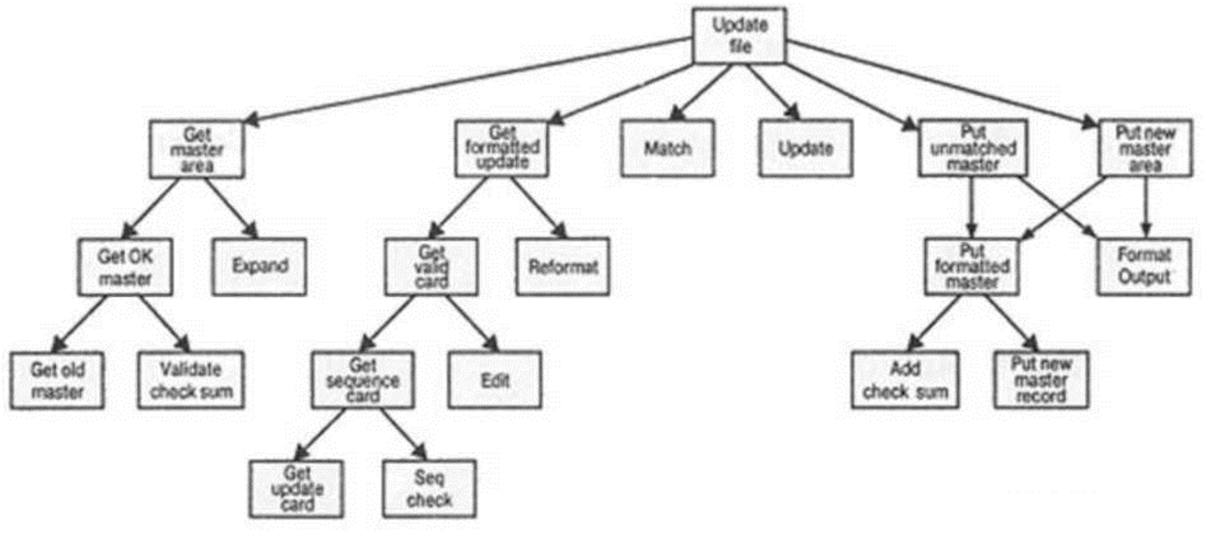
Без катлаулы системалар белән әйләндереп алынган. Мәсәлән, шәхси санак; теләсә нинди агач, чәчәк, хайван; атомнан алып йолдызлар һәм галактикаларга кадәр теләсә кайсы материя яисә җәмгыять институтлары – корпорацияләр, берләшмәләр – катлаулы система мисаллары булып торалар.

Күпчелек катлаулы системалар иерархияле структурага ия. Ләкин барлык программ тәэминат та катлаулы түгел. Бер үк кеше проектлый, төзи һәм куллана торган кушымталар классы бар. Тик аларны куллану өлкәсе чикләнгән. Корпоратив программ тәэминат төзегәндә, сәнәгать өчен программалауны караганда катлаулылык сораулары пәйда була. Программ тәэминат төзү катлаулылыгының дүрт төп сәбәбе бар:

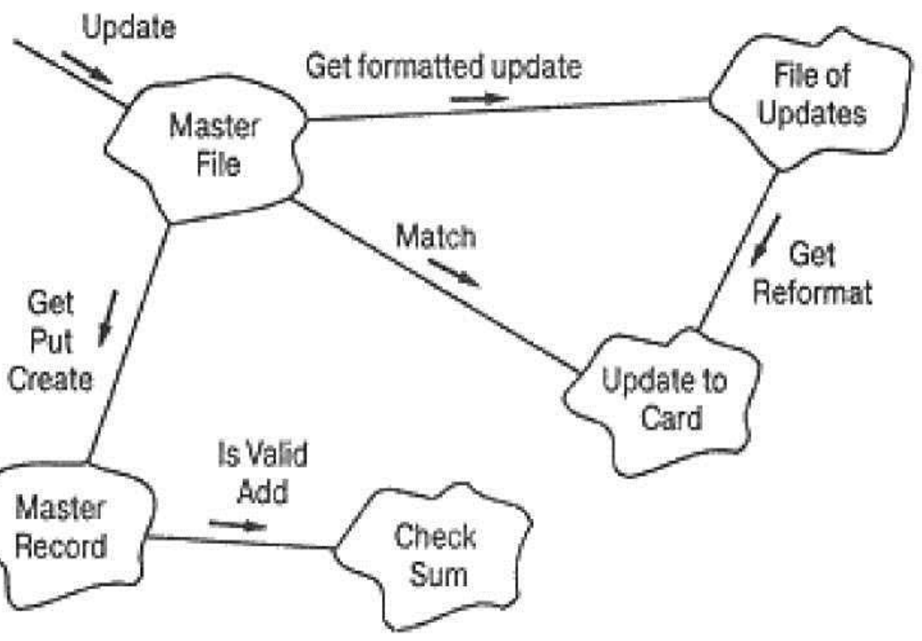
- программа төзүгә нигез булып торган предмет өлкәсенең катлаулылыгы;
- программа төзү белән идарә итүнең катлаулылыгы;
- программаның җитәрлек сыгылмалы булырга тиешлеге;
- зур дискрет системаларны тасвирлауның катлаулылыгы.

1.1. Декомпозиция

Декомпозиция – катлаулылык белән көрәшнең бер ысулы.



Рәс. 1. Алгоритмик декомпозиция мисалы



Рәс. 2. Объектка ориентлашкан декомпозиция мисалы

Системаны бәйсез астсистемаларга бүлеп, һәрберсен аерым эшләү зарур. Декомпозициянең түбәндәге ысулларын билгелиләр:

- алгоритмик декомпозиция (1 нче рәсем);
- объектка ориентлашкан декомпозиция (2 нче рәсем).

1.2. Программалау телләренең кыскача тарихы

Югары дәрәҗәле программалау телләре үсешенең түбәндәге этапларын билгелиләр:

- Беренче буын телләре (1954-1958)

FORTRAN 1 Математик формулалар

ALGOL-58 Математик формулалар

- Икенче буын телләре (1959-1961)

FORTRAN II Астпрограммалар

ALGOL-60 Блоклы структура, мәгълүмат типлары

COBOL Мәгълүматны тасвирлау, файллар белән эш

LISP Исемлекләрне эшкәртү, күрсәткечләр, чүпне җыю

- Өченче буын телләре (1962-1970)

PL/1 FORTRAN+ALGOL+COBOL

Pascal ALGOL-60 ның дәвамчысы, варисы

Simula Класслар, мәгълүматны абстракцияләү

- Буыннан буынга күчү чылбырының өзелүе (1970-1980)

C Эффектив, нәтиҗәле югары дәрәҗәле тел

FORTRAN 77 Блоклы структура, мәгълүмат типлары

- Объектка ориентлашкан программалау үсешен күрсәтүче телләр (1980-1990)

Smalltalk 80 Чиста объектка ориентлашкан программалау теле

C++ C + Simula

Ada83 Катгый типлаштыру; Pascalнең көчле йогынтысы

- Инфраструктуралар барлыкка килү (1990-...)

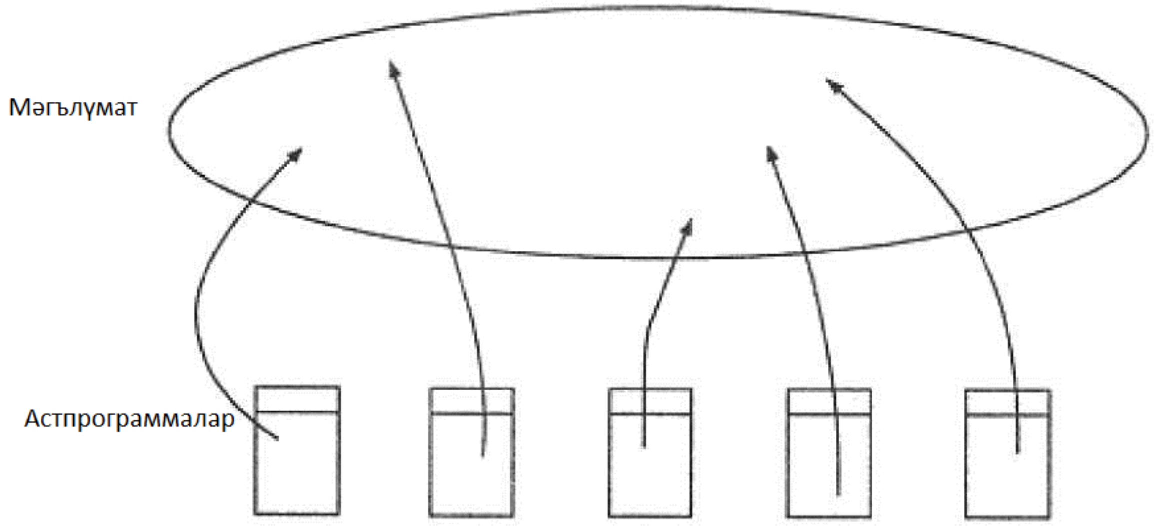
Java Блоклы структура, мәгълүмат типлары

Python Объектка ориентлашкан сценарийлар теле

Visual C# Microsoft.NET мохите өчен Java теленең конкуренты.

1 нче буын (3 нче рәсем) башлыча фәнни һәм техник исәпләүләр өчен кулланыла.

Телләр ассемблерның катлаулылыгыннан азат итәләр, бу исә санакның техник детальләренә игътибар итмәскә мөмкинлек бирә.



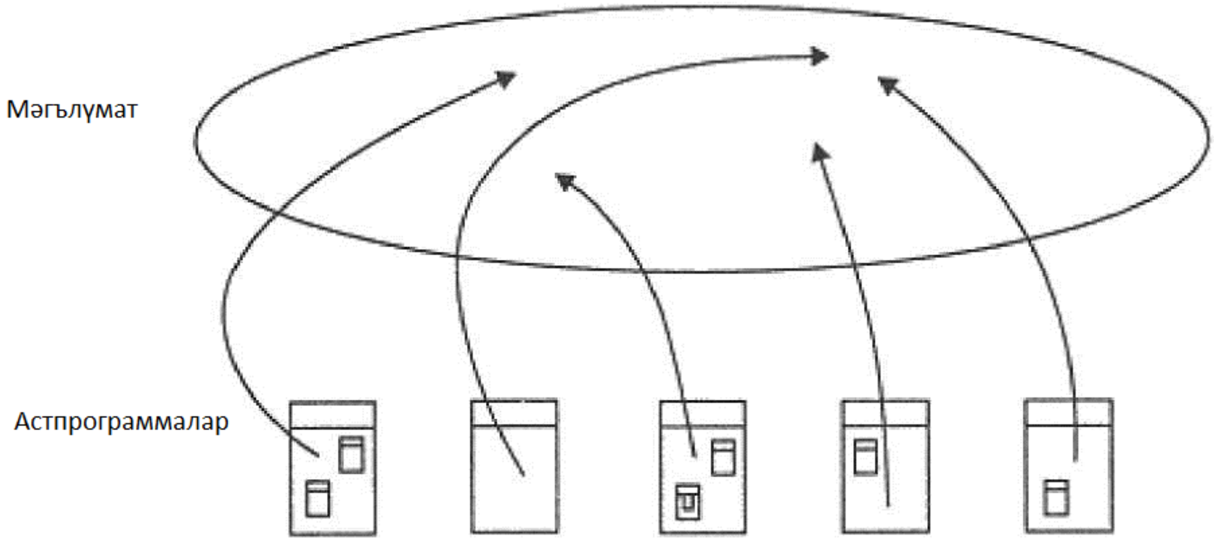
Рәс. 3. Беренче буын телләре топологиясе

Беренче буын телләрендә язылган программалар мәгълүмат һәм астпрограммалардан торучы чагыштырмача гади структурага ия.

2 нче буын телләре (4 нче рәсем) алгоритмик абстракцияләргә басым ясый, бу үзлек программа эшләүчеләрне предмет өлкәсенә якынайта. Абстракт программ функцияләрне астпрограммалар рәвешендә тасвирларга мөмкинлек бирүче процедур абстракция барлыкка килә.

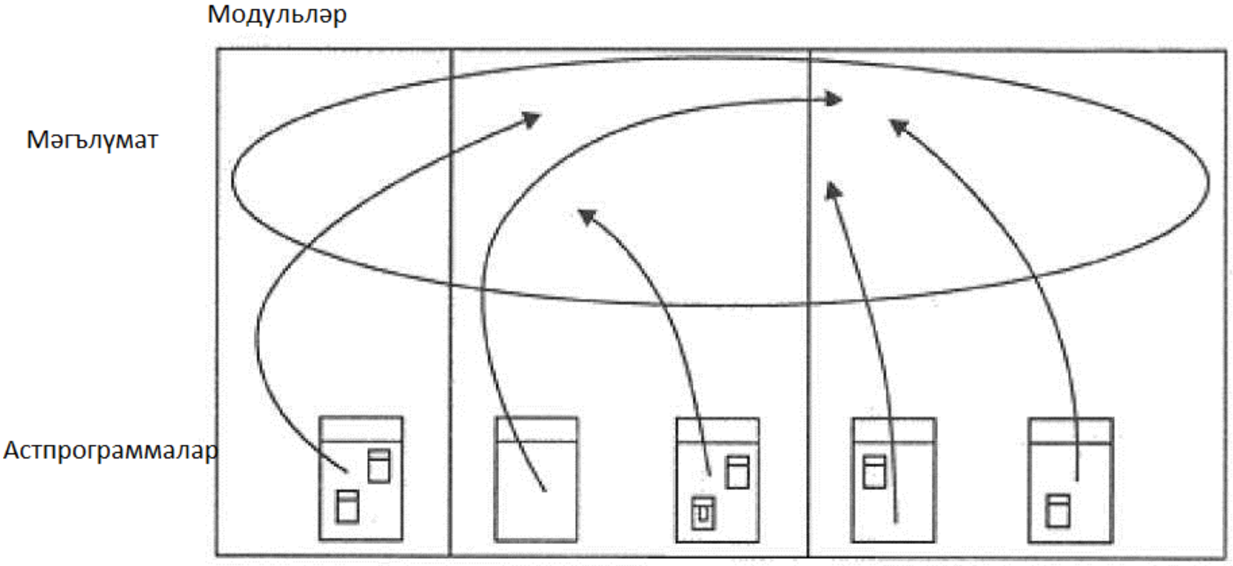
3 нче буын телләренә (5 нче рәсем) аппарат тәэминатның бәясе кискен кимүе, шул ук вакытта аның җитештерүчәнлегенең

экспоненциаль закон буенча артуы зур йогынты ясый. Телләр мәгълүматны абстракцияләргә мөмкинлек бирә һәм кулланучы үз мәгълүмат типларын тасвирлый ала.



Рәс. 4. Икенче буын телләре топологиясе

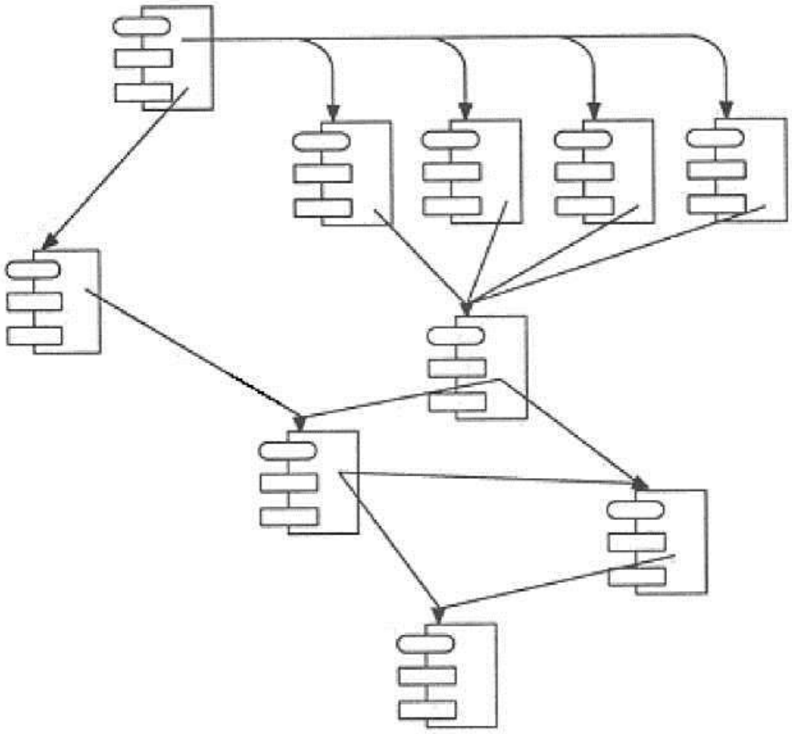
1970 нче елларда конкрет мәсьәләләрне чишү өчен берничә мең программалау теле эшләнелә, ләкин аларның күбесе юкка чыга. Вакыт сынавын узган, хәзерге вакытта билгеле булган телләр генә кала.



Рәс. 5. Өченче буын телләре топологиясе. Модульләргә бергә үзгәртеләчәк астпрограммар җыела, ләкин аларны абстракцияләүнең яңа техникасы итеп карамыйлар.

1980 нче елларда объектка ориентлашкан программалау көчле үсеш кичерә (6 нчы рәсем). Бу чор телләре программ тәэминатның объектка ориентлашкан декомпозициясен башкарырга ярдәм итә. 90 нчы елларда зур күләмдә интеграллашкан сервислар, мөмкинлекләр

тәкъдим итүче инфраструктуралар (J2EE, .NET) барлыкка килә.



Рәс. 6. Объектка ориентлашкан программалау топологиясе

Конструкциянең төп элементы булып астпрограмма түгел, логик яктан бәйле класслар һәм объектлардан торган модуль хезмәт итә.

1.3. Программалауның төп парадигмаларына кыскача күзәтү

Санак техникасының теоретик концепциясенә нигез салучы *фон-Нейман* концепциясе буенча, мәгълүматны эшкәртү өчен процессор мәгълүмат белән бер үк урында – оператив (җәһәт) хәтердә урнашкан инструкцияләрне (командаларны, күрсәтмәләрне) үти.

Шул рәвешле, мәгълүматны эшкәртү процессының төп ике асылын аерып күрсәтеп була: инструкцияләр җыелмасы булган код һәм мәгълүмат, бирелмәләр. Сайлап алынган программалау технологиясенә бәйле рәвештә барлык программалар да концептуаль яктан үзләренең кодлары яки үзләренең мәгълүмат җыелмалары тирәсендә оештырылган.

Бүгенге көн төп программалау парадигмаларын карап үтик:

1) Процесска ориентлашкан парадигма, программа бер-бер артлы үтәлүче операцияләрдән тора – фон-Нейман моделе. Бу очракта код мәгълүматка тәэсир итә. Бу парадигманы гамәлгә ашыручы телләр процедур яки императив дип атала. Мондый телләр мисалы булып С,

Pascal, һ.б. лар тора.

2) Объектка ориентлашкан парадигма, бу очракта программа мәгълүматның аерым җыелмаларын – объектларны эшкәртүче код фрагментлары берлеге итеп карала. Мондый объектлар бер-берсе белән интерфейс аша тәэсир итешәләр. Мәгълүмат кодка мөрәҗәгать итү, керү белән идарә итә.

Алгоритмның катлаулылыгы артканда процесска ориентлашкан парадигма өчен җитди проблемалар туа. Программалауның объектлы принципларына күчү программаның эчке оештырылуын яхшыртырга мөмкинлек бирә, моның нәтиҗәсе булып программ комплекслар эшләү вакытында нәтиҗәлелек арту тора.

Югарыда әйтеп үтелгән төп ике парадигма белән беррәттән хәзерге вакытта тагын ике парадигма кулланыла:

3) Аппликатив яки функциональ парадигма. Бу ысулның төп идеясе булып программа үти торган функцияне формаль яктан билгеләү тора. Шул рәвешле, кирәкле нәтиҗә алу өчен санак алар аша үтәргә тиешле халәтләр эзлеклелеген билгеләү урынына бирелгән мәгълүмат өчен кулланганда кирәкле нәтиҗә китереп чыгарырлык функцияне билгеләргә кирәк:

$$\overline{y} = f(\overline{x})$$

Бу очракта программа язу бирелгән гади стандарт функцияләрдән катлаулы функция төзүгә кайтып кала:

$$\overline{y} = f_1(f_2(f_3(\dots), f_4(\dots), \dots))$$

Бу парадигманы кулланучы телләр булып, мәсәлән, LISP, Wolfram Mathematica (Wolfram Language) һәм ML тора. Бу ысулны кулланганда мәгълүмат та, код та бертөрле структуралы тезмәләр белән күрсәтелә, димәк, программа, интерпретатор идарәсендә эшләп, үз кодын мәгълүмат кебек эшкәртә ала. Бу очракта код белән мәгълүмат арасында аерма әкренләп югала. Шуңа күрә бу парадигманың мөһим куллану өлкәләренең берсе булып ясалма интеллект (шәкли фәһем) системалары тора.

Искәрмә. Кодларны мәгълүмат эшкәрткән кебек эшкәртү процесска ориентлашкан алымны кулланганда да мөмкин, ләкин алай булганда программалау түбән дәрәҗәле мохиттә – ассемблер телендә башкарылырга тиеш.

4) Кагыйдәләр системасын куллануга нигезләнгән парадигма **(**логик программалау парадигмасы**)**. Бу алымны кулланганда программаның операторлары язылган тәртиптә түгел, рөхсәт

шартлары (русча разрешающие условия – РУ) анализы нигезендә үтәләләр.

Бу парадигмада программа парлар исемлегеннән тора:

$РУ_1 \rightarrow D_1$

$РУ_2 \rightarrow D_2$

…………

$РУ_N \rightarrow D_N$

$РУ_1$, $РУ_2$, ...,$РУ_N$ рөхсәт шартлары үтәлгәндә тиешле D_1, D_2, ... D_N гамәлләре үтәлә .

Программа эше рөхсәт шартларын циклик тикшерү һәм алар хак булганда тиешле гамәлләрне үтәүдән тора.

Логик программалау теле мисалы – PROLOG теле.

Логик программалау вакытында программаның структурасы мәгълүматны эшкәртү алгоритмын алыштырып кую берлеге итеп күрсәтүче Марков нормаль алгоритмнары теоретик концепциясе белән концептуаль бәйләнгән:

$T_{11} \rightarrow T_{12}$

$T_{21} \rightarrow T_{22}$

……………..

$T_{N1} \rightarrow T_{N2}$

Рөхсәт итүче шартлар операторлары һәм алыштырып куюлар йомгаклаучы шарт табылганчы циклик рәвештә карап чыгыла.

Без бу курста объектка ориентлашкан парадигмага бәйле сорауларны карарбыз.

2. DELPHI ДА ПРОГРАММА СТРУКТУРАСЫ

2.1. Delphi проекты

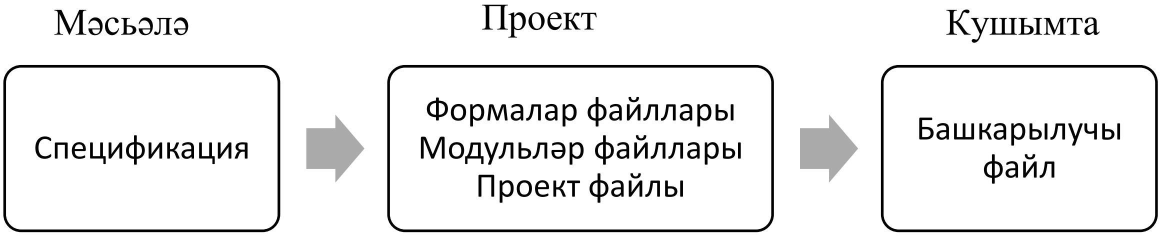
2.1.1. Проект төшенчәсе

Күпчелек программалауга өйрәтү курсларында бу сорауга игътибар аз бирелгәнлектән, проектны тәфсилләбрәк карыйк. Кушымта күп элементлардан җыела: формалар, программа модульләре, тышкы библиотекалар, рәсемнәр, пиктограммалар, һ.б. Һәр элемент аерым файлга урнаштырыла һәм катгый билгеләнү өлкәсенә ия була. Кушымта төзү өчен кирәкле файллар җыелмасы проект дип атала. Компилятор проект файлларын эзлекле рәвештә

эшкәртә һәм алардан башкарылучы файл төзи. Проектның төп файлларын берничә типка бүлеп була:

- Формаларны тасвирлау файллары – компонентлары белән формаларны тасвирлаучы DFM өстәмәле текст файллары. Бу файлларда үзлекләр тәрәзендә билгеләнгән үзлекләрнең башлангыч кыйммәтләре языла.
- Программа модульләре файллары – PAS өстәмәле текст файллары, аларда Delphi телендәге башлангыч программа кодлары урнаша. Бу файлларда формалар һәм компонентлар тарафыннан ясала (генерацияләнә) торган вакыйгаларны эшкәртү ысуллары языла.
- Проектның төп файлы – DPR өстәмәле текст файлы, анда төп программалау блогы урнашкан. Проект файлы барлык кулланылучы программа модульләрен тоташтыра, анда кушымтаны эшләтеп җибәрү өчен кирәк булган операторлар бар. Бу файлны Delphi мохите үзе төзи һәм контрольдә тота.
- Проектка болардан тыш башка файллар да керә. Алар турында мәгълүмат соңрак бирелер.

Алда әйтелгәннәр нигезендә Delphi мохитендә мәсьәлә куюдан башлап әзер башкарылучы файл алуга кадәр кушымта төзү процессын түбәндәгечә күрсәтеп була (7 нче рәсем):



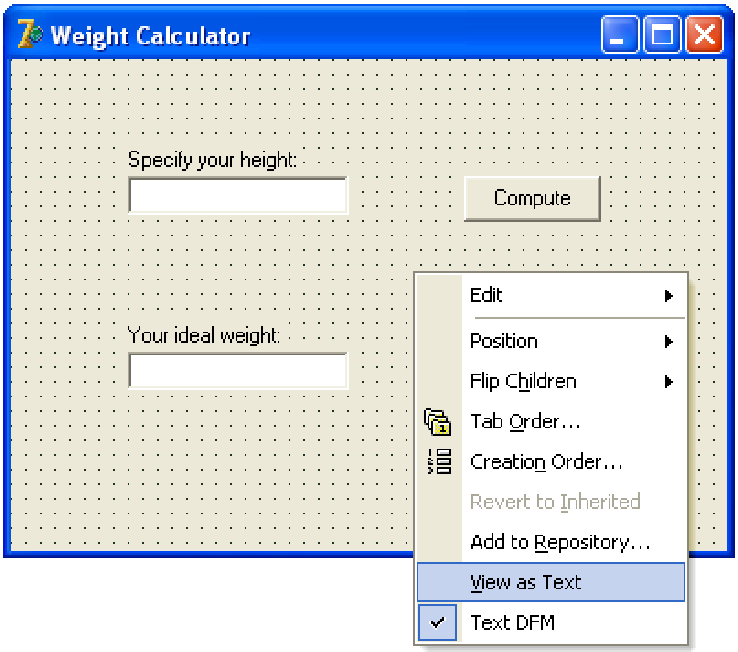
Рәс. 7. Delphi мохитендә кушымта эшләү процессы

Проект файлларының билгеләнешен һәм эчке төзелешен карыйк. Бу проектта җиңелрәк ориентлашырга ярдәм итәчәк.

2.1.2. Формаларны тасвирлау файллары

Delphi мохите белән танышу формадан башлана. Проектның беренче состав өлеше – ул форманы тасвирлаучы DFM өстәмәле текст файлы. DFM-файлда форманың һәм компонентларның кушымтаны проектлаганда үзлекләр тәрәзәсендә беренчел билгеләнгән кыйммәтләре саклана. DFM-файллар саны кушымтада кулланылучы формалар санынча. DFM-файл эчендәге мәгълүматны карыйсы

булганда, тычканның уң төймәсе белән чирттереп, форманың контекстлы сайлагын чакырырга һәм *View as Text* командасын сайларга кирәк (8 нче рәсем).



Рәс. 8. Контекстлы сайлакның View as Text командасы ярдәмендә форманы текстлы тасвирлауга күчү

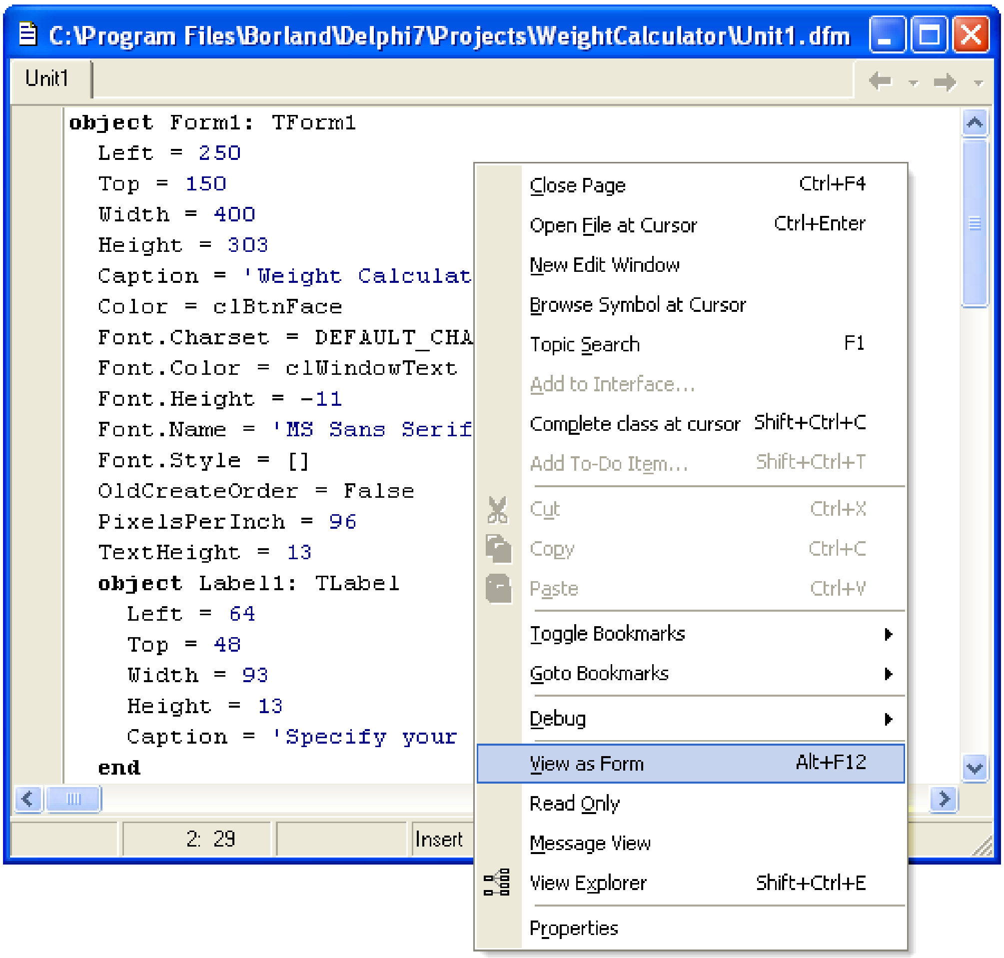
Җавап итеп Delphi мохите форманың график күрсәтелеше урынына код редакторында түбәндәге текстны бирәчәк:

```
object Form1: TForm1
  Left = 250
  Top = 150
  Width = 400
  Height = 303
  Caption = 'Weight Calculator'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 64
```

```
  Top = 48
  Width = 93
  Height = 13
  Caption = 'Specify your height:'
end
object Label2: TLabel
  Left = 64
  Top = 144
  Width = 84
  Height = 13
  Caption = 'Your ideal weight:'
end
object Button1: TButton
  Left = 248
  Top = 64
  Width = 75
  Height = 25
  Caption = 'Compute'
  TabOrder = 0
  OnClick = Button1Click
end
object Button2: TButton
  Left = 248
  Top = 160
  Width = 75
  Height = 25
  Caption = 'Close'
  TabOrder = 1
end
object Edit1: TEdit
  Left = 64
  Top = 64
  Width = 121
  Height = 21
  TabOrder = 2
end
object Edit2: TEdit
  Left = 64
```

Top = 160
Width = 121
Height = 21
TabOrder = 3
end
end

Тектсның озын булуына карамастан, аны аңлау авыр түгел. Монда махсус телдә Form1 формасының һәм аның Button1, Button2, Edit1, Edit2, Label1, Label2 компонентларының башлангыч кыйммәтләре бирелә. Аннан артыгын белү кирәк түгел, чөнки гадәттә проектлауның визуаль чаралары кулланыла, һәм форманың текстлы тасвирламасы түгел, график чагылышы (күрсәтелеше) белән эшлиләр.Текстка бернинди үзгәреш тә кертмичә, график күрсәтелешкә кайтыйк. Моның өчен код редакторының контекстлы сайлагын чакырып *View as Form* командасын сайларга кирәк. (9 нчы рәсем).



Рәс. 9. Контекстлы сайлакның View as Form командасы ярдәмендә график күрсәтелешкә күчү

Экранда яңадан форманың график күрсәтелеше (сурәте) барлыкка киләчәк. Текстка үзгәртүләр кертелгән булса, алар форманың тышкы рәвешендә чагылалар.

Форманы тасвирлау файлы (DFM-файл) проектлау этабында гына кирәк. Кушымтаны җыйганда форманың тасвирламасы DFM-файлдан үтәлүче файлның махсус өлкәсенә (ресурслар өлкәсенә) күчерелә. Кушымтаның эш вакытында форма төзелгәндә аның тасвирламасы ресурслар өлкәсеннән алына һәм форма белән компонентларны инициализацияләү өчен кулланыла. Нәтиҗәдә форма проектлаганда ничек бирелгән, экранда шулай күренә.

2.1.3. Программа модульләре файллары

Проектның һәр формасына үзенең *программа модуле (unit)* тиңдәш. Программа модулендә формага кагылышлы бөтен белдерүләр һәм вакыйгаларны эшкәртү ысуллары бар, алар Delphi мохитендә язылган. Программа модульләре PAS өстәмәле аерым файлларда урнаша. Аларның саны формалар саныннан күбрәк булырга мөмкин. Чөнки кайбер очракларда программа модульләре формаларга карамаска, ә бәлки алар эчендә ярдәмче процедуралар, функцияләр, класслар һ.б. булырга мөмкин. Мәсәлән, монда форма белән бәйле бер генә программа модуле бар. Аны карап үтик:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
```

```
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
                        {Идеаль авырлыкны хисаплау алгоритмы}
  Edit2.Text := IntToStr(StrToInt(Edit1.Text) - 100 - 10);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Close;
end;
end.
```

Программа модуле текстына кирәкле аңлатмаларны бирик. Иң башта *unit* сүзеннән соң модульнең исеме языла.

unit Unit1;

Бу исемне кулдан үзгәртергә кирәк түгел. Delphi мохите модуль исеме белән файл исеменең тәңгәл килүен таләп итә, шуңа күрә модульнең исемен үзгәртәсе булса, аны сайлакның *File | Save As...* командасын кулланып файлда яңа исем белән сакларга кирәк. Delphi мохите үзе *unit* сүзеннән соң яңа исемне куячак. Моннан соң иске модульне бетерергә кирәк.

Модульнең интерфейс бүлеге *(interface)* VCL тупламының (библиотекасының) стандарт модульләрен тоташтырудан башлана. Ул модульләрдә формада урнашкан компонентларның еш чакырылучы астпрограммалары һәм класслары билгеләнгән.

uses

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls;

Delphi мохите модульләр исемлеген программалаучы катнашыннан башка төзи һәм формага компонентлар өстәлгәндә автомат рәвештә ул исемлекне тулыландыра. Шуңа да карамастан, тоташтырылган модульләр исемлеген код редакторында кулдан үзгәртеп була.

Типлар *(type)* тасвирлау бүлегендә форма классы игълан ителгән. Килешү буенча ул *TForm1* дип атала һәм *TForm* стандарт классыннанән төзелгән (тудырылган).

```
type
 TForm1 = class(TForm)
  Button1: TButton;
  Button2: TButton;
  Edit1: TEdit;
  Edit2: TEdit;
  Label1: TLabel;
  Label2: TLabel;
  procedure Button1Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;
```

Формага урнаштырылган компонентлар форма кырлары белән күрсәтелгән. Формада алты компонент, шуңа класс тасвирламасында алты кыр. Кырлар исемнәре компонентларның үзлекләр тәрәзәсендә бирелгән исемнәре белән тәңгәл килә.

Кырлардан соң вакыйгаларны эшгәртүче ысуллар башламнары урнашкан. Һәр ысулның исемен Delphi мохите компонент исеме һәм ул тудыра (генерацияли) торган вакыйга исеме нигезендә автомат рәвештә формалаштыра. Мәсәлән, *Button1* төймәсе өчен *OnClick* вакыйгасын эшкәртү ысулы *Button1Click* дип атала.

Килешү буенча, форма компонентларын күрсәтүче кырлар һәм вакыйгаларны эшкәртүче ысуллар *published* күренүчәнлек атрибуты алалар (ул килешү буенча *TForm*ның барлык варислары өчен кулланыла). Шуның аркасында алар белән визуаль катта эшләргә, мәсәлән, исемнәрен үзлекләр тәрәзәсендә күрергә мөмкин. Delphi

мохите *published* бүлеге белән үзе идарә иткәнгә күрә, бу бүлекне код редакторында кулдан үзгәртергә кирәк түгел, визуаль кораллар (инструментлар): компонетлар палитрасы һәм үзлекләр тәрәзәсен кулланырга кирәк. Түбәндәгеләрне истә тоту зарур:

- формага компонентлар урнаштырылганда Delphi мохите форма классы тасвирламасына тиешле кырларны үзе өсти, компонентлар формадан алынганда, мохит класс тасвирламасыннан кырларны бетерә;
- формада яки компонентларда вакыйга эшкәртүчеләр билгеләнгәндә Delphi мохите үзе класста тиешле ысулларны билгели, вакыйга эшкәртүче ысуллардан барлык код алып атылганда (бетерелгәндә), Delphi мохите үзе ысулларны да бетерә.

Уңайлык өчен форма классында алдан ук буш *private* һәм *public* бүлекләре игълан ителгән, аларда теләсә нинди ярдәмче кырлар, ысуллар һәм үзлекләрне урнаштырырга мөмкин. Delphi мохите аларны «күрми», шуңа күрә алар белән программа коды катында гына эшләп була. *private* бүлегенә форманың үзенә генә кирәк атрибутларны, *public* бүлегенә башка форма һәм модульләргә дә кирәк атрибутларны урнаштырып була.

Классны тасвирлаганнан соң форма объектының үзе игълан ителә:

```
var
  Form1: TForm1;
```

Form1 үзгәрешлесе – ул проектның төп файлы, DPR-файлда төзелгән *TForm1*классы объектына сылтама.

Шуның белән модульнең интерфейс бүлеге бетә һәм реализация бүлеге *(implementation)* башлана. Башта анда форма тасвирлау файлы тоташтырыла:

```
{$R *.dfm}
```

Бу директива DFM өстәмәле барлык файлларны да тоташтырмый. Бу модульнең формасы тасвирланган бер генә DFM-файл тоташтырыла. DFM-файлның исеме йолдызчыкны директива язылган модуль исеменә алмаштырудан килеп чыга.

Аннары вакыйгаларны эшкәртүче ысуллар реализацияләнә. Алар өчен буш шаблон, урынны Delphi мохите форма классына башламнар өстәгән вакытта үзе булдыра. Программалаучы аларны тутыра.

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
            { Идеаль авырлыкны хисаплау алгоритмы}
  Edit2.Text := IntToStr(StrToInt(Edit1.Text) - 100 - 10);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Close;
end;
```

Искәрмә. Әгәр вакыйганы эшкәртү ысулын бетерергә һәм аңа сылтамаларны алып атарга кирәк булса, ысулны буш итеп калдырырга – комментарийларны, локаль үзгәрешлеләрне игълан итүне, барлык язылган кодны бетерергә кирәк. Проектны саклаганда яки компиляцияләгәндә Delphi мохите үзе тексттан буш ысулларны алып атачак.

Модульнең башлангыч текстын игътибар белән караганда ачыкланмаган сорау кала: формадагы төймәләргә басканда *Button1Click* һәм *Button2Click* ысулларын чакыру ничек тәэмин ителә, модуль текстында бу турыда искә дә алынмый. DFM-файл ны карыйк. Үзлекләрнең кыйммәтләрен билгеләүдән тыш анда вакыйгаларны эшкәртүчеләрне билгеләү дә бар.

```
object Button1: TButton
  ...
  OnClick = Button1Click
end
object Button2: TButton
  ...
  OnClick = Button2Click
end
```

Бу тасвирлау форма ысулларын тиешле вакыйгаларга бәйләргә мөмкинлек бирә. Сүз уңаенда, Delphiдагы вакыйгаларның чынлыкта үзлекләр булуын, тик аларның кыйммәтләре булып ысулларга сылтамалар торганын истә тотсак, китерелгән фрагментның мәгънәсе яхшырак аңлашыла. Шулай итеп, вакыйгаларны билгеләү форма үзлекләрен билгеләүдән күпкә аерылмый, асылда алар бер үк әйбер.

2.1.4. Проектның төп файлы

Компилятор проектта нинди файллар булуын белсен өчен, ниндидер оештыручы башлам кирәк. Һәм ул чынлап та бар. Ул DPR (Delphi Project) өстәмәле проект файлы. Бу файл төп программа файлы булып тора, ул *uses* операторы ярдәмендә проект составындагы барлык модульләрне тоташтыра. Һәр проект өчен бер генә DPR-файл була.

File | New | Application командасы буенча яңа кушымта төзелә башлаганда Delphi мохите автоматик рәвештә проект файлы булдыра. Яңа формалар өстәлә барганда бу файл автомат рәвештә эчтәлеген, рәвешен үзгәртә. Эш төгәлләнеп проект компиляция өчен әзер булганда DPR-файлда компиляторга тапшырылачак программа модульләренең исемлеге булачак. DPR-файлның эчен, эчтәлеген күрү өчен Delphi мохите сайлагында *Project | View Source* командасын сайларга кирәк. Код редакторында түбәндәге текст белән яңа бит барлыкка киләчәк:

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Forms модулендә *Application* объекты билгеләмәсе булганга, бу модульне тоташтыру барлык программалар өчен дә мәҗбүри. Бу объект теләсә ниди график кушымта нигезендә ята һәм аңа кушымта эше барышында мөрәҗәгать итеп була.

Аның артыннан тоташтырылучы Unit1 модулендә форма билгеләмәсе бар. Форма исеме фигуралы җәяләр эчендә китерелә. *in* директивасы модуль проектның кирәкле өлеше булуын һәм Delphi телендәге башлангыч текст рәвешендә торуын күрсәтә.

*{$R *.res}* директивасы башкарылучы файлга ресурсларны, бу очракта кушымта билгесен тоташтыра. Бу билге мәсьәләләрнең идарә панелендә күренәчәк.

Алга таба *Application* объектының өч ысулын чакыруны эченә алган программа блогы урнаша. *Initialize* ысулын чакыру кушымтаны эшкә әзерли, *CreateForm* ысулы *Form1* формасын эшкә куша һәм инициализацияли, *Run* ысулы форманы активлаштыра һәм кушымтаны эшләтә башлый. Чынлыкта *Run* ысулының эш вакыты – ул кушымтаның эш вакыты. *Run* ысулыннан чыгу кулланучы кушымтаның төп формасын япканнан соң үтәлә, нәтиҗәдә кушымта ябыла.

Искәрмә. DPR-файлны кулдан үзгәртмәскә кирәк. Бу эшне Delphi мохите башкара. Модульләрне өстәү һәм бетерү, формалар төзү белән идарә итү мохитнең командалары һәм диалог (аралашу) тәрәзәләре ярдәмендә башкарыла.

2.1.5. Проектның башка файллары

Югарыда проектның төп файллары каралды. Алардан тыш өстәмә файллар була:

- DOF өстәмәле файл (Delphi Options File), анда программалаучы тарафыннан бирелгән компиляция һәм җыю параметрлары саклана;
- DSK өстәмәле файл (Desktop), анда Delphi мохитенең бу проект өчен көйләнмәләре саклана. Delphi мохите көйләнмәләрен DSK-файлда сакласын өчен сайлакта *Tools* | *Environment Options...* командасын сайларга һәм *Environment Options* диалог тәрәзенең *Preferences* кыстырмасында *Autosave options* төркемендә *Project Desktop* пунктын билгеләргә кирәк.
- CFG өстәмәле файл (Configuration), анда компиляторның консоль варианты өчен көйләнмәләр саклана.
- DCI өстәмәле файл (Delphi CodeInsight), анда Delphi мохите программ «суфлер» (CodeInsight) өчен ясалган көйләнмәләрне саклый.
- DCT өстәмәле файл (Delphi Component Templates), анда компонентларның башлангыч үрнәкләре саклана.
- DMT өстәмәле файл (Delphi Menu Templates), анда сайлакның башлангыч үрнәкләре саклана.
- DRO өстәмәле файл, анда компонентлар саклагычының көйләнмәләре һәм өстәмәләре саклана.
- TODO өстәмәле файл – программалауга биремнәр һәм кыска искәрмәләр саклау өчен язу дәфтәре.

• DDP өстәмәле файл (Delphi Diagram Portfolio), анда компонентлар арасындагы бәйләнешләрне күрсәтүче график схемалар саклана.

• RES өстәмәле ресурслар файлы (RESource). Анда, мәсәлән, мәсьәләләрнең идарә панелендә сурәтләнә торган кушымта билгесе саклана.

Проектка логик яктан автоном булган элементлар керә ала: нокталы рәсемнәр (BMP-файллар), билгеләр (ICO-файллар), белешмә файллары (HLP-файллар) һ.б., тик алар белән программалаучы үзе идарә итә.

Хәзер проект составын чагылдыра торган рәсемне аныклап була (10 нчы рәсем):

Рәс. 10. Delphi мохитендәге проект составы

Проектның составы аңлашыла. Аның белән идарә итү – проектны төзү һәм саклау, модульләрне өстәү һәм бетерү, компиляция параметрларын билгеләү, кушымтаны җыю һәм эшләтеп җибәрү мәсьәләләрен карыйк.

2.2. Проект белән идарә итү

2.2.1. Проектны төзү, саклау һәм ачу

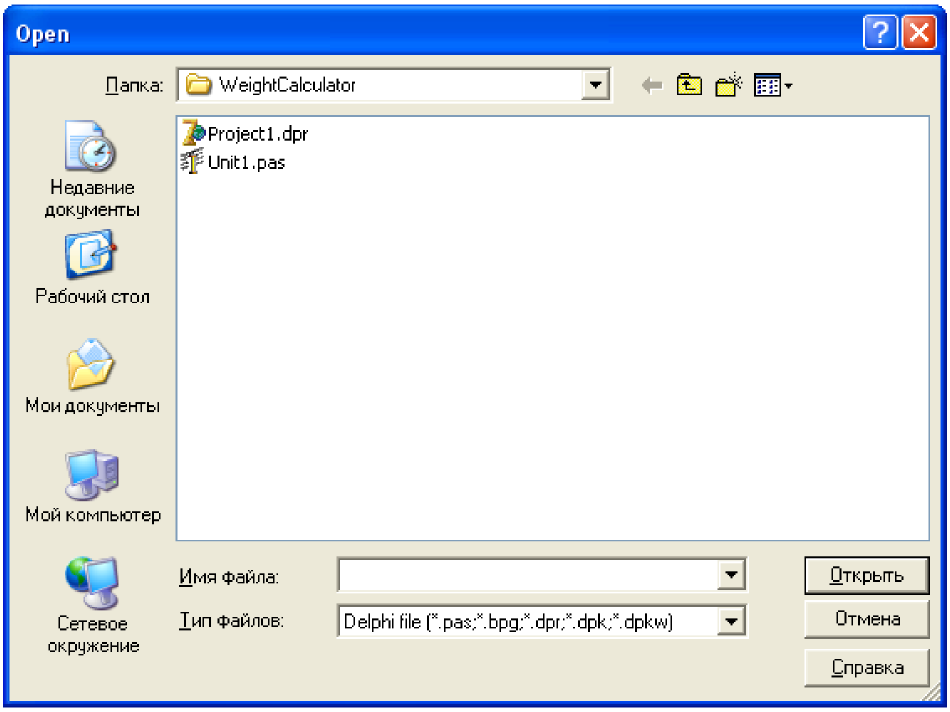
Delphi мохите эшләтеп җибәрелгәндә автоматик рәвештә яңа проект төзелә. Бу программалаучыга уңайлы булсын өчен шулай

эшләнгән. Delphi мохитен яңадан эшләтеп җибәрмичә яңа проект төзисе булса, сайлакның *File* | *New* | *Application* командасын үтәргә кирәк. Нәтиҗәдә иске проект ябылачак, аның урынына яңа проект төзеләчәк. Яңа проектка Delphi мохите һәрвакытта да чиста форма урнаштыра.

Кушымта өстендә эш вакытында формага яңа компонентлар өстәлә, вакыйгаларны эшкәртүчеләр языла, оның формалар өстәлә, кыскасы, кушымта проектлау өстендә эш бара. Берникадәр эш башкарылгач, проектны саклау зарур. Моның өчен төп сайлакның *File* | *Save All* командасын үтәргә кирәк. Мохит башта формалы программа модуленә исем, аннары проект өчен исем сораячак. Әгәр дискта андый исемле файл булса, Delphi мохите моның турында хәбәр итәчәк һәм булган файлны яңадан күчереп язу яки яңа исем куеп язуны расларга сораячак.

Әгәр дә проект исемен башка исемгә алыштырырга кирәк булса, сайлакның *File* | *Save Project As...* командасын кулланырга кирәк. Әгәр модуль исемен алыштырырга кирәк булса, *File* | *Save As...* командасын куллану зарур.

Delphi мохитендә элегрәк дискта сакланган проектны ачарга кирәк булганда, *File* | *Open...* командасын үтәү җитә. Экранга диалог тәрәзәсе чыгачак (11 нче рәсем), анда йөкләнүче проектның каталогын һәм исемен күрсәтергә яки исемлектән сайлап алырга кирәк булачак.



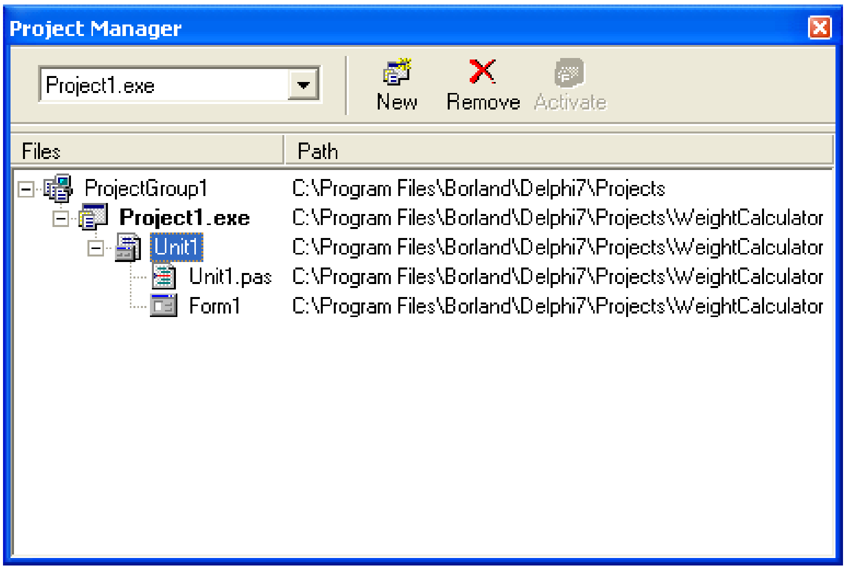
Рәс. 11. Проектны сайлау тәрәзәсе

Ачылган проект белән эшне дәвам итеп: төзәтеп, компиляцияләп, башкарып, саклап була.

Мөһим! Һәр проект өчен аерым папка төзү эшне җиңеләйтә!

2.2.2. Проект белән идарә тәрәзе

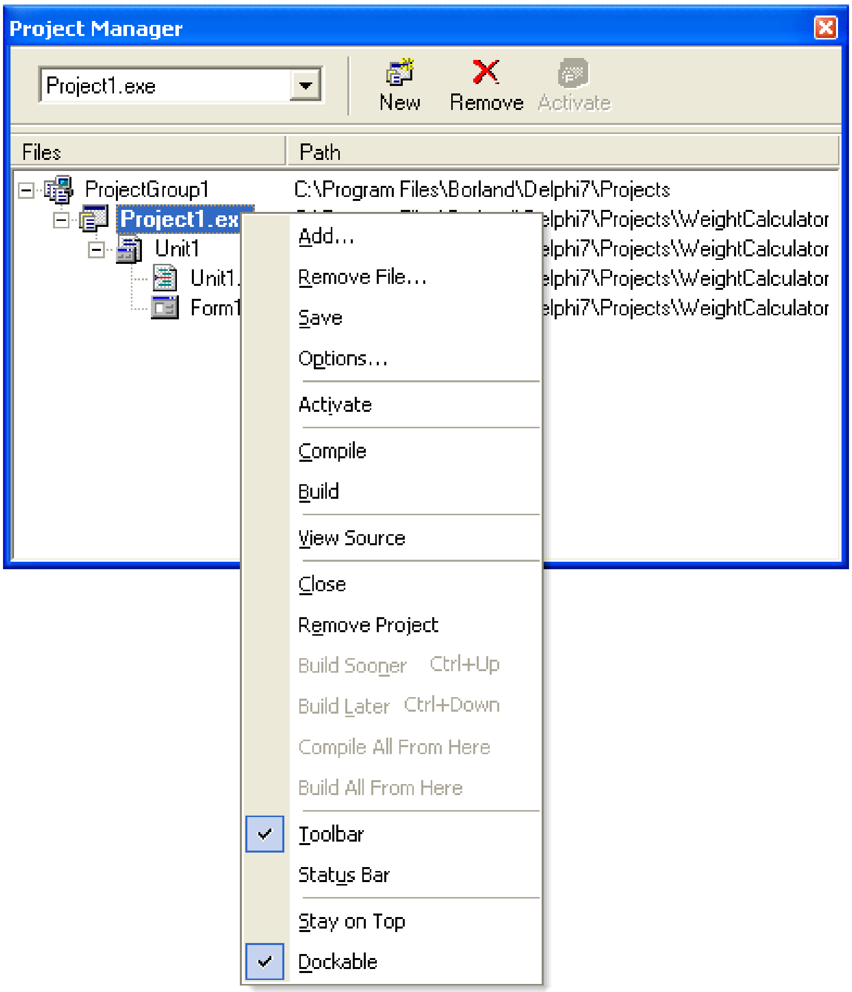
Теләсә нинди – катлаулымы, түгелме проект төзегәндә програмалаучы һәрвакытта да кайсы стадиядә булуын белергә, проект составын күз алдына китерергә, тиешле файлны эшләтеп җибәрә алырга, проектка берәр яңа файл өстәргә яки кирәк түгелен алып атарга, компиляция параметрларын билгели алырга һ.б. тиеш. Моның өчен Delphi мохитендә проект белән идарә итү тәрәзе – Project Manager бар. Чынлыкта ул проектның төп файлын үзгәртү өчен визуаль инструмент. Проект белән идарә итү тәрәзе төп сайлакның View | Project Manager командасы ярдәмедә чакырыла. Экранда проект агач рәвешендә күрсәтелгән тәрәзә барлыкка килә. (12 нче рәсем).



Рәс. 12. Проект белән идарә итү тәрәзәсе

Калын шрифт белән аерып алынган Project1 элементы – ул безнең проект. Аның исеме башкарылучы файл исеме белән тәңгәл. Башкарылучы файл проектның бөтен модульләрен җыю һәм компиляцияләү нәтиҗәсендәтөзелә. Барлык модульләр исемлеге бәйле элементлар рәвешендә күрсәтелә. Элемент форма булып торса, ул, үз чиратында ике бәйле элементтан тора: форманың программа модуле (PAS-файл) һәм форманы тасвирлау (DFM-файл).

Проект белән идарә итү Project1 элементында тычканның уң төймәсен чиртеп чакырылган контекстлы сайлак ярдәмендә башкарыла. (13 нче рәсем).



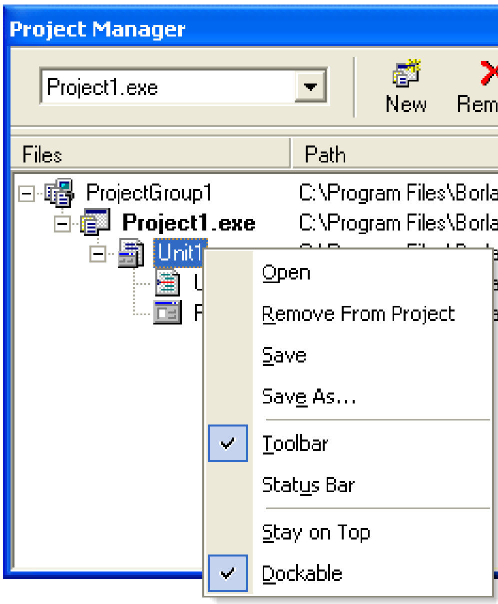
Рәс. 13. Проектның контекстлы сайлагы

Контекстлы сайлакның командалары билгеләнеше түбәндәге таблицада кыскача китерелгән:

Команда	Тасвирлама
Add…	Проектка булган файлны (модульне) өсти.
Remove File…	Файлны (модульне) проекттан алып ата, бетерә.
Save	Проектны дискка саклый.
Options…	Проектны көйләү тәрәзәсен чакыра (Project Options).
Activate	Проектны актив итә (проектлар төркеме белән эшләгәндә).
Close	Проектны яба.
Remove Project	Проектны төркемнән алып ата (бетерә).
Build Sooner	Проектны җыю чиратын билгеләүче исемлек буенча өскә күчерә. Проектлар төркеме белән эшләгәндә кулланыла.

Build Later	Проектны җыю чиратын билгеләүче исемлек буенча аска күчерә. Проектлар төркеме белән эшләгәндә кулланыла.
Compile All From Here	Үзгәртелгән проектларны билгеләнгән, аерып алынган проекттан башлап, җыя, компиляцияли. Проектлар төркеме белән эшләгәндә кулланыла.
Build All From Here	Барлык проектларны билгеләнгән, аерып алынган проекттан башлап, җыя, компиляцияли. Проектлар төркеме белән эшләгәндә кулланыла.

Аерым модуль белән идарә итү тиешле элементта, мәсәлән, Unit1 дә тычканның уң төймәсен чиртеп чакырыла торган контекстлы сайлак ярдәмендә башкарыла. (14 нче рәсем).



Рәс. 14. Проект белән идарә итү тәрәзәсендә модульнең контекстлы сайлагы

Контекстлы сайлакның төп командалары билгеләнеше түбәндәге таблицада кыскача китерелгән:

Команда	Тасвирлама
Open	Модульне ача. Модуль составында форма булса, экранга аның график күрсәтелеше чыга. Алай булмаса, экранга программа модуленең башлангыч тексты белән код редакторы чыга.

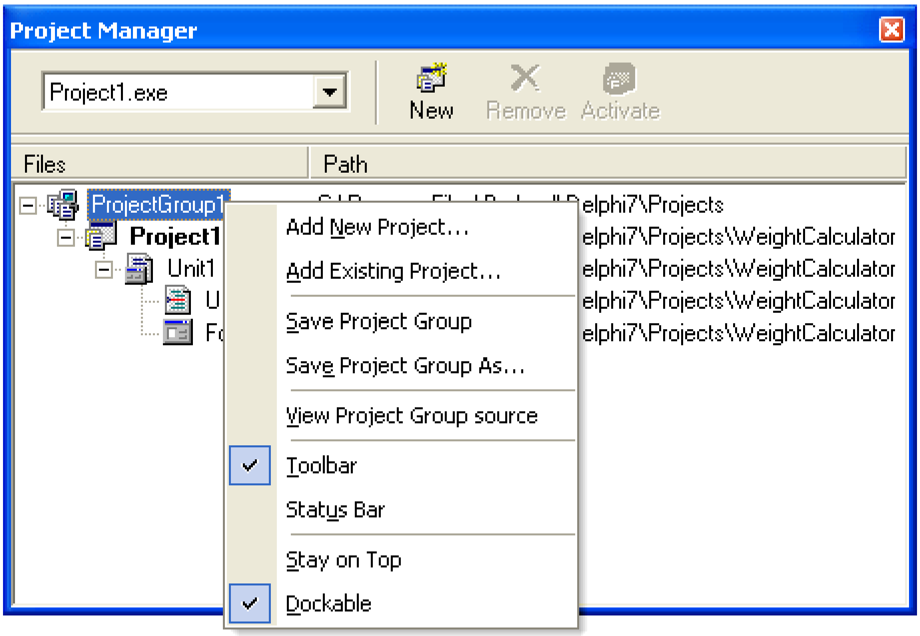
Remove From Project	Модульне проекттан алып ата, бетерә.
Save	Модульне дискка саклый.
Save As…	Модульне яңа исем белән саклый.

Шулай итеп, һәрвакытта да проектның нинди файллардан торуын белеп була, ә аның белән идарә итү авырлык тудырмаячак.

2.2.3. Проектлар төркеме

Практикада берничә проект үзара логик бәйләнгән булырга мөмкин, мәсәлән, динамик ялганучы туплам (библиотека) проекты белән бу туплам кулланылучы кушымта проекты бәйле. Delphi мохите мондый проектларны төркемгә берләштерергә мөмкинлек бирә. Нәкъ шуның өчен проект белән идарә итү тәрәзәсендә ProjectGroup1 тамыр элементы бар, аның буйсынган элементлары булып логик яктан бәйле проектлар тора. Элементларның тәртибе проектларны җыю тәртибен билгели. Тәртипне Build Sooner һәм Build Later командалары ярдәмендә үзгәртеп була. Бу командалар тиешле проектта тычканның уң төймәсе белән чиртеп чакырыла торган контекстлы сайлакта урнашкан.

Хәзергә төркемдә бер генә проект – Project1 бар. Төркемгә башка проектлар өстәү өчен ProjectGroup1 элементында тычканның уң төймәсе белән чиртеп чакырыла торган контекстлы сайлакны кулланырга мөмкин. (15 нче рәсем).



Рәс. 15. Проектлар төркеменең контекстлы сайлагы

Контекстлы сайлакның командалары билгеләнеше кыскача түбәндәге таблицада китерелгән:

Команда	Тасвирлама
Add New Project…	Яңа проект төзи һәм аны төркемгә өсти.
Add Existing Project…	Булган проектны төркемгә өсти.
Save Project Group	Проектлар төркемен тасвирлый торган файлны саклый.
Save Project Group As…	Проектлар төркемен тасвирлауны башка исемдәге файлда саклый.
View Project Group source	Проектлар төркемен тасвирлаучы текст файлын күрсәтә.

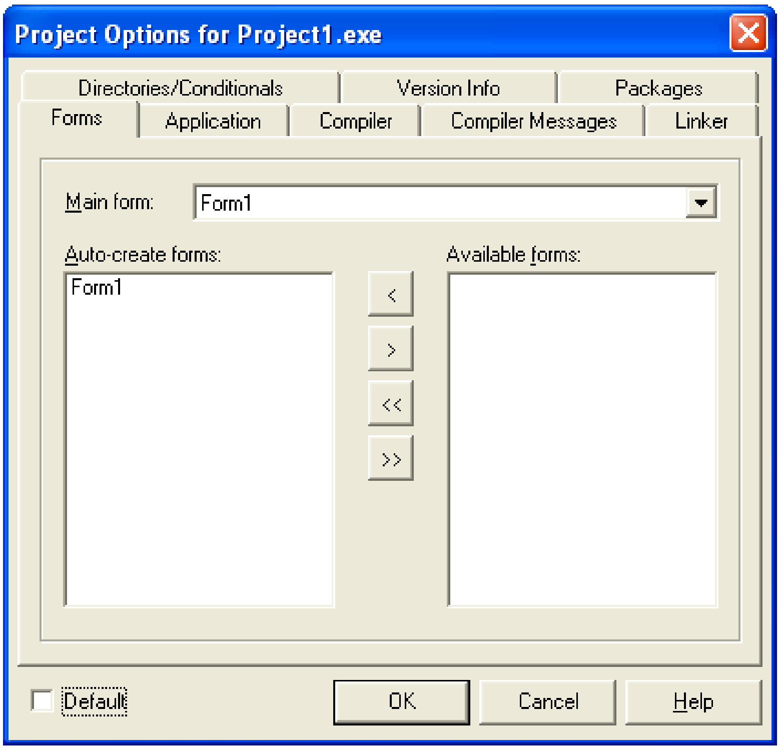
Төркемгә берничә проект берләшкәндә Delphi мохите бу төркем тасвирламасы белән махсус текст файлы төзи. Файл BPG өстәмәле (Borland Project Group), аның исеме кулланучыдан сорала. Бер генә проекттан торучы төркемнәр өчен BPG-файл төзелми.

2.2.4. Проектның параметрларын көйләү

Проектның күп төрле параметрлары була, аларны кулланып программалаучы кушымтаны компиляцияләү һәм җыю процессы белән идарә итә. Проектның параметрларын Project Options тәрәзәсендә билгеләп була. Моның өчен төп сайлакта *Project | Options...* командасын сайларга яки проект белән идарә итү тәрәзәсендә Project1 элементының контекстлы сайлагын чакырырга һәм Options... командасын сайларга кирәк. Экранга диалог тәрәзәсе чыга, анда параметрларның тиешле кыйммәтләрен генә куясы кала.

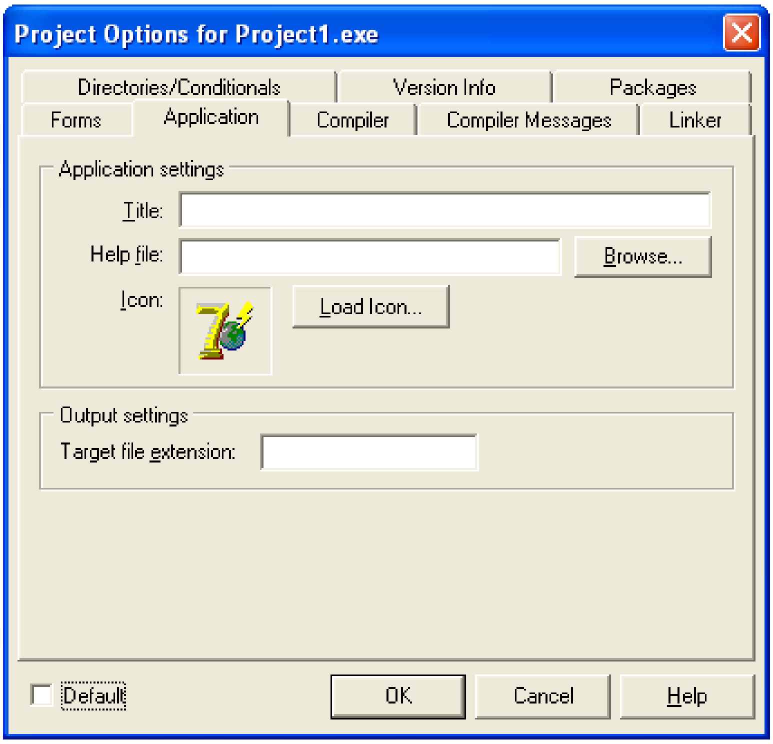
Проектның параметрлары диалог тәрәзәсендә берничә кыстыргыч бар. Параметрлар бик күп, шуңа еш куллана торганнарын гына карап үтик.

Forms кыстыргычында (16 нчы рәсем) кушымтаның төп формасын (Main form) билгеләргә һәм Auto-create forms исемлегендә төп форма белән бер вакытта төзеләчәк формаларны күрсәтергә мөмкин.



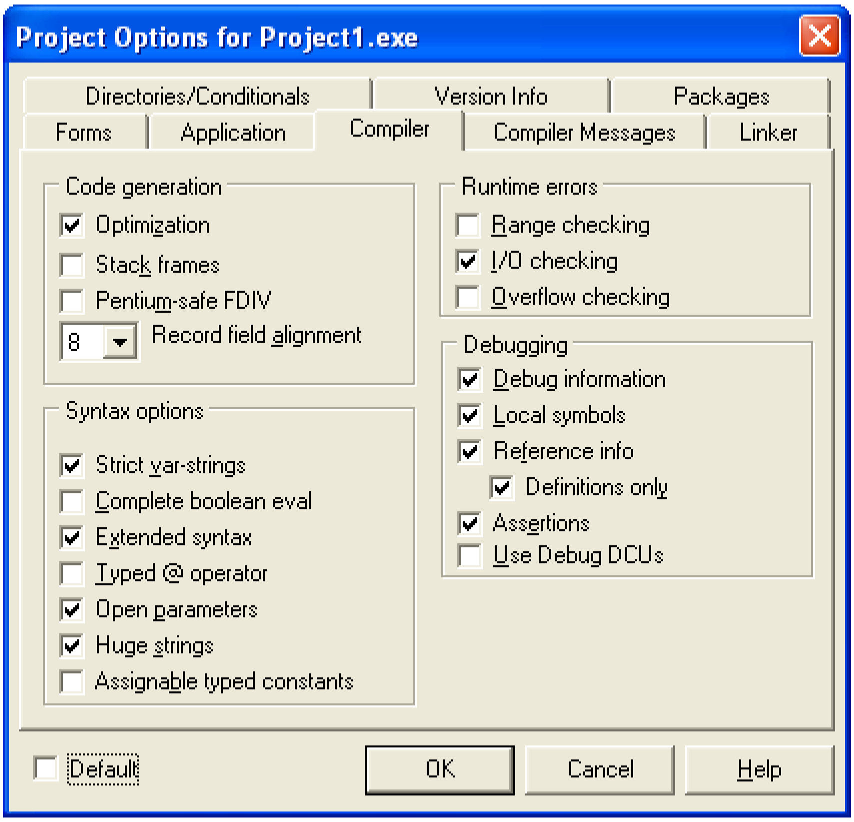
Рәс. 16. Проект параметрлары тәрәзәсе. Forms кыстыргычы

Application кыстыргычында (17 нче рәсем) программаның исемен (Title) билгеләргә мөмкин. Delphi мохитендә өстәмә рәвештә белешмә файлы (Help file) һәм билге (Icon) биреп була.



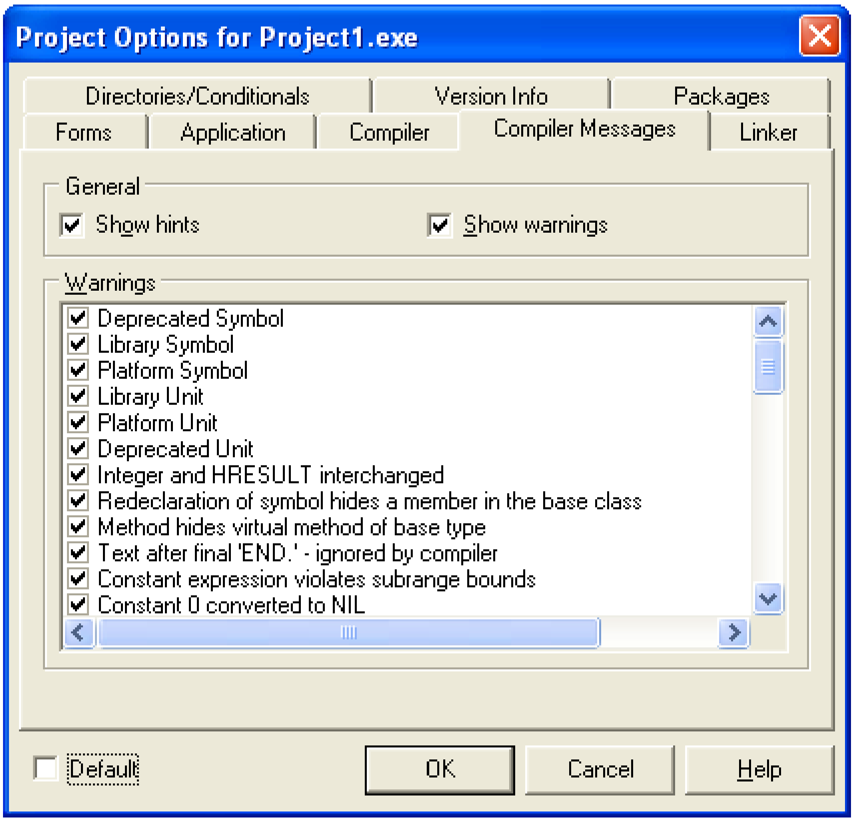
Рәс. 17. Проект параметрлары тәрәзәсендә Application кыстыргычы

Compiler кыстыргычында компилятор параметрлары көйләнә. Аларның иң кызыклылары булып Optimization (генерацияләнүче кодның оптимизациясен тоташтыра) һәм Use Debug DCUs (систем тупламнарның башлангыч кодларын дөресләргә, көйләргә мөмкинлек бирә) бәйле беркетелгән төймәләре тора. Алар икесе дә программаны көйләгәндә файдалы: беренчесен өзәргә, икенчесен ялгарга кирәк.



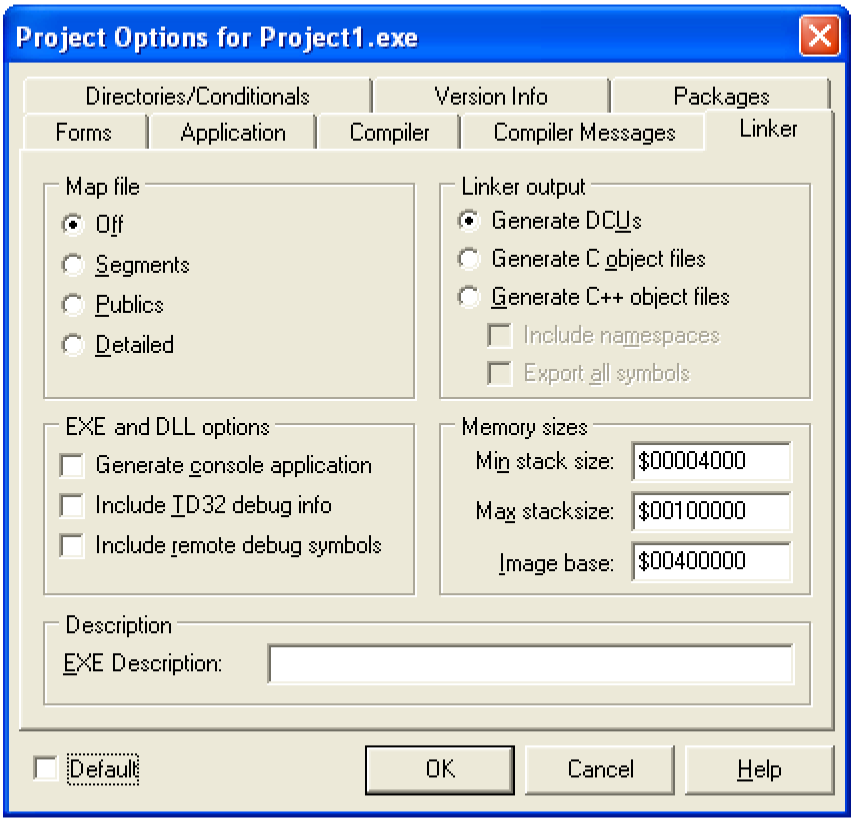
Рәс. 18. Проект параметралы тәрәзәсендә Compiler кыстыргычы

Compiler Messages кыстыргычында (19 нчы рәсем) компиляторның шикле кодка сизгерлеге көйләнә. Show hints һәм Show warnings бәйле беркетелгән төймәләр ялганган булса, компилятордан хаталар турында хәбәрләр генә түгел, ә бәлки файдалы киңәшләр (hints) һәм кисәтүләр (warnings) алырга була.



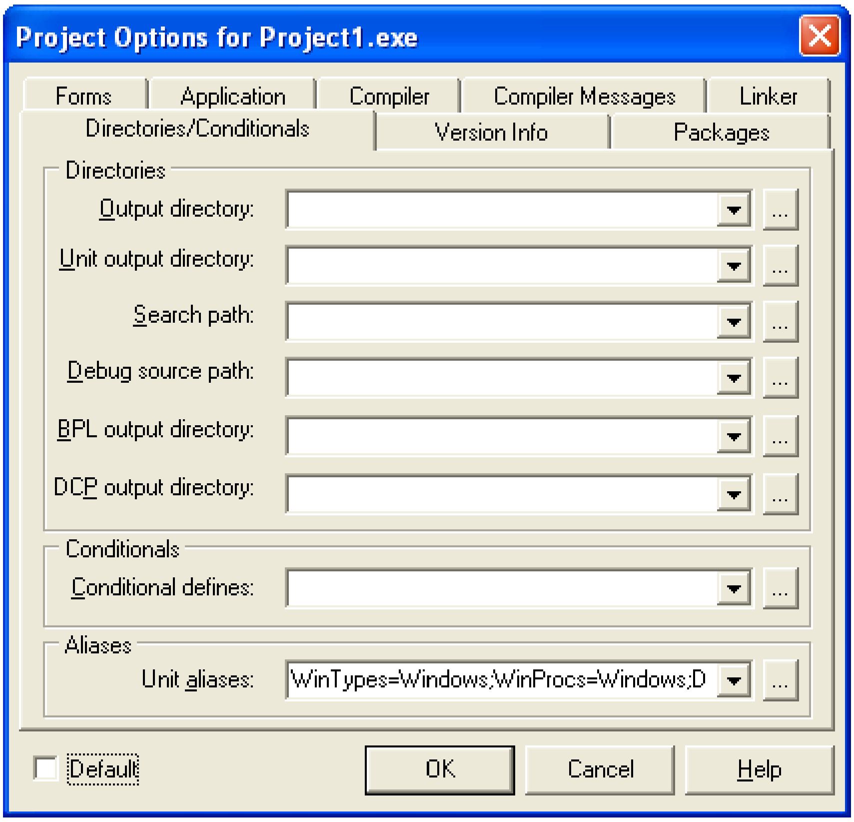
Рәс. 19. Проект параметрлары тәрәзәсендә Compiler Messages кыстыргычы

Linker кыстыргычында (20 нче рәсем) проектны җыю параметрлары көйләнә. Delphi мохитендә эшләүчеләргә Memory sizes төркеменә, аеруча Min stack size һәм Max stack size параметрларына игътибар итәргә кирәк. Алар гамәли программа стэгенең минималь һәм максималь үлчәмнәрен куялар. Рекурсив астпрограммаларны актив кулланучы кушымта төзегәндә бу параметр кыйммәтләрен арттырырга кирәк булуы бар.



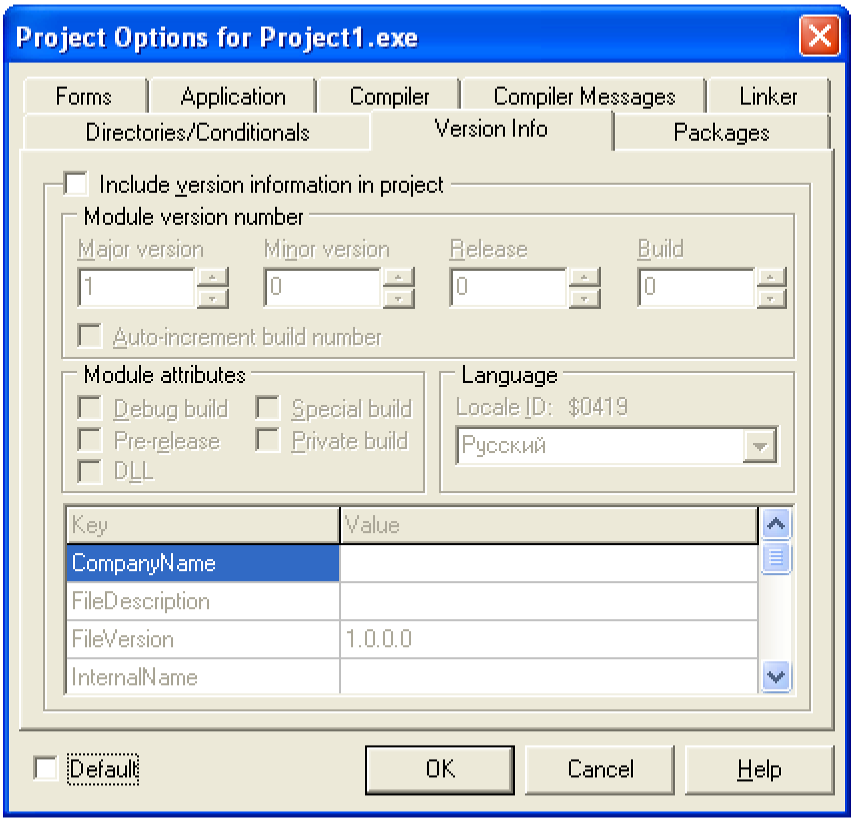
Рәс. 20. Проект параметрлары тәрәзәсендә Linker кыстыргычы

Directories/Conditionals кыстыргычында (21 нче рәсем) төрле файллар өчен каталоглар билгеләп була. Аларның аеруча мөһимнәре: Output directory – башкарылучы файл урнашачак каталог; Unit output directory – вакытлы (арадаш) объект модульләре (DCU-файллар) урнашачак каталог; Search path – программа модульләрен эзләү башкарылачак каталоглар исемлеге.



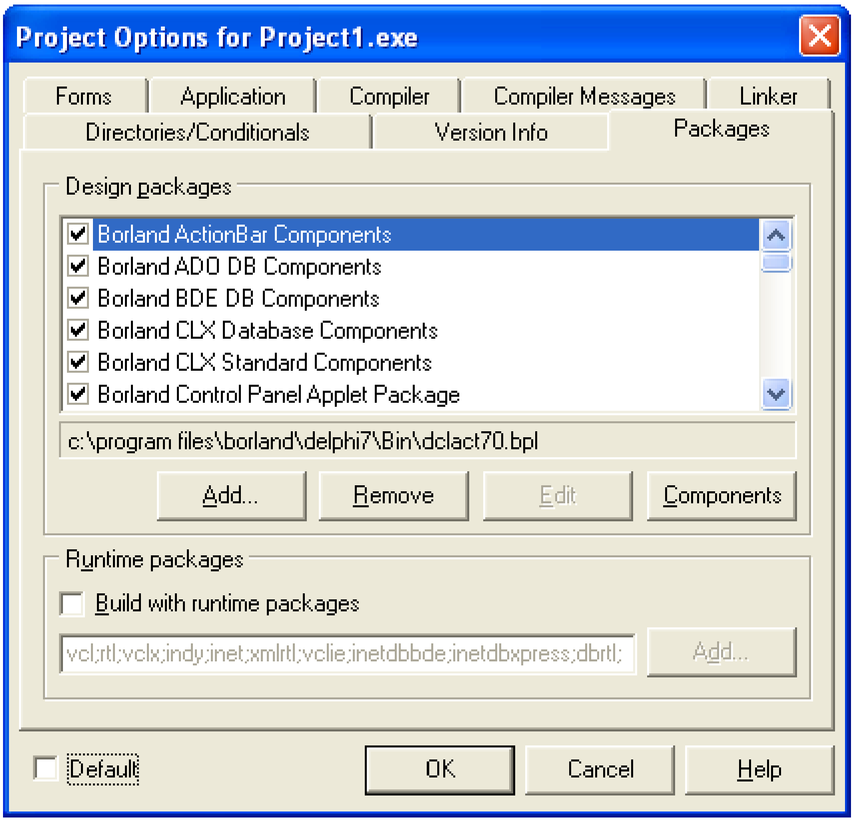
Рәс. 21. Проект параметрлары тәрәзәсендә Directories/Conditionals кыстыргычы

Version Info кыстыргычында (22 нче рәсем) кушымтаның версиясе турында мәгълүмат чыгарыла. Бу мәгълүмат башкарылучы файлга урнашсын өчен Include version information in project төймәсен ялгарга кирәк. Версия номеры дүрт сан белән бирелә: Major version – программа версиясенең өлкән номеры (аны программага концептуаль яңа мөмкинлекләр керткәндә арттыралар); Minor version – программа версиясенең кече номеры (аны гадәттә программаның функциональ мөмкинлекләрен аз гына киңәйткәндә арттыралар); Release – программаның көйләнгән һәм заказчы тарафыннан кулланугa әзер версиясенең номеры; Build – проектны җыюның тәртип номеры (ул Auto-increment build number төймәсе ялганган булса, проект компиляцияләнгәндә автомат рәвештә арта). Барлык бу параметрлар кирәкле мәгълүмат алу өчен кирәк, алар программа эшенә йогынты ясамыйлар. Тик версия турындагы мәгълүмат махсус урнаштыру программалары тарафыннан тупламнарның (библиотекаларның) яңарак версияләре иске версияләргә алышынмауны контрольләү өчен кулланылырга мөмкин.



Рәс. 22. Проект параметрлары тәрәзәсендә Version Info кыстыргычы

Packages кыстыргычында (23 нче рәсем) проектта кулланылучы пакетлар исемлеге белән идарә итәргә мөмкин. Пакетлар – ул компонентларның тышкы тупламнары (библиотекалары). Build with runtime төймәсе башкарылучы файлның үлчәмен киметергә мөмкинлек бирә. Ул компонентларның кодын башкарылучы файлга урнаштыру урынына компонентларның тышкы тупламнарын куллану исәбенә килеп чыга. Бу режим зур күләмдәге уртак компонентлар нигезендә төзелгән берничә программа өстендә эшләгәндә кулай.



Рәс. 23. Проект параметрлары тәрәзәсендә Packages кыстыргычы

Проектның барлык параметрлары куелгач, аның компиляциясен башлап була.

2.2.5. Проектны компиляцияләү һәм җыю

Проектны компиляцияләү һәм җыю проект белән эшнең теләсә кайсы стадиясендә башкарыла алалар. Компиляция дип программ модульләрнең башлангыч текстларыннан (PAS-файллардан) объект модульләрен (DCU-файлларны) булдыруны атыйлар. Җыю дип объект модульләрдән башкарылучы файлны булдыруны атыйлар. Delphi мохитендә компиляция һәм җыю берләштерелгән.

Компиляция башкару өчен сайлакның *Project* | *Compile <Проект исеме>*командасын сайларга яки төймәләрнең (клавишларның) Ctrl+F9 комбинациясенә басарга кирәк. Бу очракта соңгы компиляциядән соң үзгәртелгән барлык башлангыч модульләр компиляцияләнә. Нәтиҗәдә һәр программа модуле өчен DCU өстәмәле (Delphi Compiled Unit) файл төзелә. Аннан соң Delphi мохите проектның төп файлын компиляцияли һәм DCU-модульләрдән башкарылучы файл җыя, аның исеме проект исеме белән тәңгәл килә. Delphi мохите компиляторы башкарылучы файлдан кулланылмаган программа кодын алып ата,

шуңа тоташтырылучы модульләрдәге артык объектлар һәм астпрограммалар турында борчылмаска була.

Компиляция һәм җыюның аерым төре була – проектның башлангыч текстларга мөрәҗәгать итә, кирә алган программа модульләрен тулы мәҗбүри компиляцияләп, аннары башкарылучы файлны җыю. Бу очракта аларга соңгы компиляциядән соң үзгәрешләр кертелгәнме, юкмы икәне мөһим түгел. Проектны тулы компиляцияләү төп сайлакның *Project* | *Build* *<Проект исеме>* командасы ярдәмендә башкарыла. Нәтиҗәдә шулай ук башкарылучы файл төзелә, тик моңа вакыт күбрәк китә.

2.2.6. Әзер кушымтаны эшләтеп җибәрү

Күпсанлы компиляцияләрдән соң барлык хаталар да төзәтелеп башкарылучы файл булдырылгач, хезмәтнең нәтиҗәсенә карап була. Моның өчен төзелгән кушымтаны сайлакның *Run* | *Run* командасы яки F9 клавишы ярдәмендә эшләтергә кирәк. Эшләтү алдыннан компиляция процессы проектка үзгәртүләр кертелгән булса автоматик рәвештә кабатланачак һәм ул беткәннән соң кушымта эшли башлаячак. Нәтиҗәдә экранга төп форма чыгачак.

2.2.7. Модуль структурасы

Delphi һәм Delphi 2.0 программалау мохитләре чыкканнан соң кушымталар эшләү технологиясе үзгәрде, нәтиҗә буларак «модуль» (unit) конструкциясенең әһәмияте үсте, аның синтаксисына кайбер үзгәртүләр кертелде. Болар түбәндәге әйберләрдә чагыла:

- Delphi мохитендә һәр формага үз модуле тиңдәш һәм кагыйдә буларак визуаль булмаган алгоритмик гамәлләр дә аерым модуль булып оештырылалар. Шулай итеп, Delphi мохите Object Pascal телендәге модульле программалауны дәвам итә, бу исә модульләрне киң кулланып программалауның прогрессив һәм ышанычлы ысулын куллануга стимул булып тора.
- Object Pascal дә модульнең яңа бүлеге – Borland (Turbo) Pascal һәм Delphi ның 16 разрядлы версиясендә булмаган finalization тәмамлау бүлеге кертелгән. Бу бүлек кагыйдә буларак кушымтага инициализация бүлегендә бирелгән ресурсларны азат итү өчен кулланыла һәм кушымтаның дөрес, «чиста» тәмамлануын гарантияли.
- Borland (Turbo) Pascal версиясендә кулланылган begin сүзеннән аермалы буларак, Object Pascal дә инициализация модулен билгеләү өчен махсус initialization сүзе кулланыла.

Модульле программалауның төп принцибы булып бүлеп идарә итү принцибы тора. Модульле программалау – ул программаны модуль дип аталучы зур булмаган бәйсез блоклар җыелмасы итеп оештыру, аларның төзелеше һәм тәртибе билгеле кагыйдәләргә буйсына.

«Модуль» сүзенең кулланылышы төрле булырга мөмкин икәнен искәртеп китик: программалау телләренең синтаксик конструкциясе (Object Pascal дә unit) һәм зур программаны аерым блокларга бүлү (алар процедура, функция рәвешендә реализацияләнә ала).

Модульле программалауны куллану программаны сынауны һәм хаталар табуны җиңеләйтә. Аппаратка бәйле мәсьәләләр башка мәсьәләләрдән катгый бүленә-аерыла алалар, бу исә төзелүче программаларның мобильлеген яхшырта.

Программаларның эффектлылыгын күтәрү процессы гадиләшә, чөнки кирәкле модульләрне башкаларыннан бәйсез рәвештә күп тапкыр үзгәртеп карап була. Моннан тыш модульле программаларны аңлау күпкә җиңел, модульләр башка программаларны төзүче блоклар буларак куллана алалар.

«Модуль» термины программалауда программа төзегәндә модуль принципларын кертә башлаганнан кулланыла. 70 нче елларда модуль дип билгеле кагыйдәләр буенча язылган берәр процедура яки функцияне аңлыйлар. Мәсәлән, «Модуль гади, ябык (бәйсез), күренүчән (50 дән 100 юлга кадәр), мәсьәләнең бер генә функциясен реализацияли торган, бер керү һәм бер чыгу нокталы булырга тиеш». Тик гомуми таләпләр булмый һәм еш кына модуль дип 50 юлдан кыскарак булган теләсә кайсы процедураны атыйлар.

Беренче тапкыр программа модуленең төп сыйфатларын күпмедер дәрәҗәдә ачык итеп Д. Парнас (Parnas) формалаштыра: «Бер модульне язу өчен икенчесенең тексты турында минималь мәгълүмат белү җитәргә тиеш». Шулай итеп, Парнас беренче булып программалауда мәгълүматны яшерү (information hiding) концепциясе турында әйтә.

Парнас билгеләмәсе буенча модуль булып иерархиянең иң түбән катында да, башка процедуралар-модульләр чакыру гына башкарылган иң югары катында да урнашкан теләсә кайсы аерым процедура (функция) тора ала.

Тик 70 нче елларда кулланылган процедура һәм функция кебек синтаксик конструкцияләр генә мәгълүматны ышанычлы яшерүне

тәэмин итә алмый, чөнги аларга катлаулы программаларда үзләрен ничек тотасын еш кына алдан әйтеп булмый торган глобаль үзгәрешлеләр тәэсир итә. Бу проблеманы глобаль үзгәрешлеләр тәэсиренә бирешми торган яңа синтаксис конструкциясе төзеп кенә хәл итеп булган. Мондый конструкция төзелә һәм модуль дип атала.

Беренче тапкыр махсус конструкция 1975 елда Н.Вирт тарафыннан тәкъдим ителә һәм аның яңа Modula теленә кертелә. Шул ук елда Modula теленең тәҗрибә реализациясе эшләнә. Берникадәр үзгәртүләрдән соң яңа тел тулысынча 1977 елда реализацияләнә һәм Modula-2 исеме ала. Соңарак, шуңа ошаган, берникадәр аермалары булган конструкцияләр башка программалау телләренә дә кертеләләр: Pascal Plus (Уэлш һәм Бастард, 1979 ел), Ada (1980), Turbo Pascal нең 4.0 версиясе һәм башкалар.

Баштарак катлаулы программ комплекслар реализацияләгәндә модуль процедуралар һәм функцияләр белән беррәттән аерым мәсьәләнең реализация детальләрен берләштерүче һәм ышанычлы яшерүче конструкция булырга тиеш дип карала. Әмма Borland (Turbo) Pascal телендә модульнең барлык теоретик мөмкинлекләре дә кулланылмый. Мәсәлән, бу телдә эчке модульләргә булышлык юк (эчке процедуралар һәм функцияләр кебек), импорт (uses) башка модульләрдән объектларны сайлап импортларга мөмкинлек бирми. Бу хәл, шулай ук персональ санаклар чыга башлаганнан соң программалаучы кешеләр саны кисәк арту (ә бу программалаучыларның теоретик хәзерлекләрен түбәнәйтә) телнең Object Pascalгә кадәр кулланылган версиясендә кушымталар эшләгәндә модульләр процедура һәм функцияләрнең тупламнарын (библиотекаларын) төзү чарасы буларак куллануга китерә. Иң яхшы әзерлекле программалаучылар гына бу тел конструкциясенең бөтен көчен проектларын структуралаштырганда куллана алалар. Object Pascal телендә модульнең күрсәтеп үткән чикләүләре кала. Delphi мохитендә һәр формага катгый рәвештә үз модуле тиңдәш булганга һәм визуаль булмаган алгоритмик гамәлләр кагыйдә буларак шулай ук модуль рәвешендә оештырылганга, «модуль» конструкциясен төрле квалификацияле программалаучылар аның беренчел билгеләнүе буенча куллана башлыйлар.

Шулай итеп, проекттагы модульләр саны куелган мәсьәләне бәйсез кечерәк мәсьәләләргә декомпозицияләү белән билгеләнергә тиеш. Ахыр чиктә, проект кодына теләсә нинди үзгәрешләр керткәндә

ул башкаручы локаль гамәл программаның башка бүлекләре тәэсиреннән бәйсез булуы кирәк булганда, модуль бер процедура кую өчен дә файдаланырга мөмкин. Мәсәлән, модульне болай куллану реаль вакыт мәсьәләләре өчен хас. Аларда программаның ышанычлылык һәм алдан күреп булу критерийлары мөһим.

Модульле программалау куллану белән бәйле тагын бер теоретик сорау карыйк. Бу сорау модуль проектының формасына карый.

Модульле проектның иерархик структурасын уңай формага китерү аны эшләү процессын яхшыртырга мөмкинлек бирә. Бирелгән модуль үзенә тоташтыручы модульләр саны һәм аны тоташтыручы модульләр саны проектның катлаулылыгына тәэсир итә. Йодан (Yourdon) бирелгән модульдән тоташтырылучы модульләр санын модульләр белән идарә итү киңлеге дип атый. Зур идарә итү киңлеге белән берләттән бик кечкенә яки бик зур идарә итү киңлеге модульләргә бүлүнең начар схемасы күрсәткече булып тора. Гомуми очракта идарә итү киңлеге 10 нан артмаска тиеш. Бу сан «тылсымлы» 7 саны белән бәйле, ул психология нигезләмәләренә, аеруча мәгълүмат «кисәкләре» («chunking») теориясенә корылган. Кешенең кыска вакытлы хәтерендә мәгълүмат «кисәкләре» саклау мөмкинлекләре чикле. Психологик экспериментлар безнең кыска вакытлы хәтердә 5-11 (уртача 7) «кисәк» саклануын күрсәттеләр. Хәтер берүк вакытта 7 «кисәк» мәгълүмат белән эш итә ала. Бу чик үткәндә кеше күбрәк ялгыша. Мәгълүматны тиешле кисәкләргә бүлеп оештыру кешенең кыска вакытлы хәтерен эффектив куллану һәм материалны яхшырак аңлау өчен мөһим гамәл. Кешеләр күп очракта мондый бүлүне үзеннән-үзе башкаралар. Ә программалаучы модульләр белән идарә итү киңлеген 7 дән күпкә арттырмыйча үзенә үзе булыша ала.

Delphi мохитендә программалауга килгәндә, модульле проект формасы турындагы киңәшләрне программалаучы төзегән модульләргә карата кулланырга кирәк. Чөнки Delphi кодның формаларны эшкәртү белән бәйле төп өлешен автомат рәвештә генерацияли, һәм программалаучыга стандарт тоташтырыла торган модульләр турында артык уйлыйсы юк.

Моннан тыш, бу принципларны класслар иерархиясен проектлаганда да кулланырга кирәк. Мәсәлән, Object Pascalнең алдан билгеләнгән класслар иерархиясендә ике классның, TObject һәм Exceptionның гына 7 дән күпкә артык туры варис класслары бар. Моны TObject классының глобаль нигез роле һәм Exception

классының «саналмалы» характеры белән аңлатып була.Тагын өч классның мондый саны 8-9 тирәсе, ә башка классларның турыдан туры варислары 7 дән ким.

Модульләрне кулланганда аларның исемен дөрес күрсәтергә кирәк. Стандарт модульләрне тоташтырганда аларның идентификаторларын uses та дөрес язу җитә.

Модульләр төзегәндә кайбер үзенчәлекләрне истә тотарга кирәк:

- бер үк вакытта бер исемле модульләрне куллану рөхсәт ителми;
- башламда (unit) күрсәтелгән модуль идентификаторлары башлангыч (.pas) һәм объектлы (.dcu) кодлы файллар исемнәре белән тәңгәл килергә тиеш.

Түбәндә модульнең гомуми структурасы китерелгән, ул һәр бүлекнең мәгънәсен һәм билгеләнешен аңлатучы комментарийлар белән тулыландырылган.

unit <МодульИдентификаторы>;

{Интерфейс бүлеге.}

interface

{Бу бүлектә бирелгән модульнең башка кулланучы төзегән һәм стандарт модульләр белән, шулай ук төп программа белән үзара бәйләнеше тасвирлана. Башкача әйткәндә – модульнең «тышкы дөнья» белән бәйләнеше.}

{Интерфейс бүлегенең импорт исемлеге.}

uses

{Бу исемлектә интерфейс бүлекләрендәге мәгълүматка бу модульдән мөрәҗәгать итеп, кереп була торган модульләрнең идентификаторлары өтер аша санала. Монда мәгълүматлары бу модульнең interface бүлегенең тасвирламаларында кулланыла торган модульләрнең идентификаторларын китерү кирәк. }

{Интерфейс бүлегенең экспорт исемлеге. Ул модульдә билгеләнгән, тик аларны куллану бу модуль исемен uses юлына керткән барлык башка модульләр һәм программаларда рөхсәт ителгән константалар, типлар, үзгәрешлеләр, процедура һәм функция башламнарын тасвирлау бүлекчәләреннән тора. Процедура һәм

функцияләр өчен формаль параметрлары тулысынча тасвирланган башламнар китерелә. }

const
type
var
procedure
function

{Реализация бүлеге.}

implementation

{Бу бүлектә модуль тасвирламасының реализация (шәхси) өлеше күрсәтелә. Аңа башка модуль һәм программалар мөрәҗәгать итә алмый. Башка сүзләр белән әйткәндә – модульнең «эчке кухнясы».}

{Реализация бүлегенең импорт исемлеге.}

uses

{Бу исемлектә интерфейс бүлекләрендәге мәгълүматка бу модульдән мөрәҗәгать итеп була торган модульләрнең идентификаторлары өтер аша санала. Монда түбәндәге шартларны канәгатьләндерүче барлык кирәкле модульләрнең идентификаторларын күрсәтү кирәк: алардан алган мәгълүмат Interface бүлеге тасвирламаларында кулланылмаган булырга һәм аларны куллану турында башка модульләр белмәскә тиеш.}

{Модульнең үзенең эчке тасвирламалары өчен бүлекчәләр.}

label
const
type
var
procedure
function

{Бу бүлекчәләрдә модульнең үзе тарафыннан башкарыла торган алгоритмик гамәлләрне тасвирлаучы һәм бары тик шушы модульнең «шәхси милке» булып торган тамгалар, константалар, үзгәрешлеләр, процедуралар һәм

функцияләр тасвирлана. Бу тасвирламалар башка модульләргә ябык. Бүлекчәдәге процедуралар һәм функцияләрнең башламнарын формаль параметрлары исемлекләреннән башка күрсәтергә ярый. Әгәр дә башламнар параметрлары белән күрсәтелгән булса, формаль параметрлар исемлеге тиңдәш процедура (функциянең) interface бүлегендәге шундый ук исемлеге кебек булырга тиеш. }

{Инициализация бүлеге.}

initialization

{Бу бүлектә махсус initialization һәм finalization сүзләре арасында модульне дөрес эшләтеп җибәрү өчен кирәкле башлангыч көйләү операторлары урнаша. Программада кулланылучы модульләрне инициализацияләү бүлекләре операторлары uses та нинди тәртиптә язылган булсалар, программаны башлангыч эшләтеп җибәргәндә шул ук тәртиптә башкарылалар. Инициализация операторлары кирәк булмаса, махсус initialization сүзен төшереп калдырып була.}

finalization

{finalization тәмамлау бүлегенең булуы мәҗбүри түгел, ул initialization инициализация бүлеге белән бергә генә була ала. Тәмамлау бүлегендә модульнең эше тәмамланганда (ул гадәттә кушымтаның эше беткәндә була) башкарылачак операторлар исемлеге урнаша. Кушымта модульләренең finalization бүлекләре бу модульләрнең initialization бүлекләрен башкару тәртибенә кире тәртиптә башкарылалар. Тәмамлау бүлеге гадәттә кушымтага инициализация бүлегендә бирелгән ресурсларны азат итү, чистарту өчен кулланыла, һәм шуның белән кушымтаның эшен дөрес, «чиста» бетерүне гарантияли. Бу бигрәк тә кушымтаның эше гадәттән тыш булган очраклар килеп чыгу аркасында туктатылганда мөһим.}

end.

Object Pascalдә модульнең инициализация бүлеген билгеләү өчен

махсус initialization сүзе кулланыла, Borland (Turbo) Pascal дә begin сүзе кулланылды. begin сүзен куллану тыелмый (элекке версияләр белән күчүчәнлекне саклау), тик киңәш ителми.

finalization бүлеге Object Pascalнең Delphi 2.0дән башлап кергән яңалыгы. Бу бүлек Borland (Turbo) Pascalдә булмады.

Модульләр белән эш вакытында аларның процедуралар һәм функцияләрдән төп аермаларын истә тотарга кирәк.

Глобаль һәм локаль үзгәрешлеләрнең эш өлкәләренең традицион кагыйдәләре модульләр өчен үтәлми. «Модуль» тел конструкциясе төп программада игълан ителгән глобаль үзгәрешлеләрнең модульнең эчке тасвирламаларына тәэсирен бетерү өчен махсус эшләнә.

Шуңа күрә, әгәр дә барлык блоклар өчен дә ачык тасвирламалар (глобаль тасвирламалар) кертергә кирәк булса, модульләр өчен моны бер ысул белән генә эшләп була: махсус глобаль белдерүләр модуле төзергә (мисалда UGlob) һәм аны аның тасвирламасы кирәк булган барлык модульләрнең импорт исемлекләренә кертергә.

Башлангыч стандарт халәте ун элементтан торган исемлек структурасы белән эшне реализацияләү өчен «модуль» конструкциясен куллану мисалын карыйк. Бу мисалда модульләр арасындагы бәйләнешләргә хас структура күрсәтелгән.

unit UGlob;

{Глобаль тасвирламалар модуле}

interface

type

TVal = Integer;

{Мисалны кирәксез детальләр белән тутырмас өчен глобаль TVal тибын тасвиралау өчен типны яңадан билгеләү башкарылган.}

implementation

end.

unit UStack;

{Библиотека (туплам) модуле, үз эченә «стэк» типлы исемлек структурасы белән эш өчен кирәк сервис процедуралар һәм функцияләрне ала .}

interface

{Импортланучы модульләр исемлеге.}

uses UGlob;

{ UGlob модулен күрсәтү җитә. Чөнки interface бүлеге тасвирламаларында башка стандарт модульләр һәм кулланучы модульләреннән алынган мәгълүмат кулланылмый. Interface та тасвирланган модульләр implementation га да ачык булачак.}

{Экспорт процедуралары исемлеге.}

procedure Push (Val : TVal) ;

{Элементны стәккә кертү процедурасы.}

procedure Pop (var Val : TVal) ;

{ Элементны стәктән алу, чыгару процедурасы.}

function GetStatus : string;

{Стәкнең халәт юлын кайтаручы функция.}

implementation

{Модульнең «шәхси» импорт исемлеге.}

uses

{UStack ны реализацияләү өчен implementation бүлегендә башка модульләрне тоташтыру таләп ителми. UGlob модуленнән мәгълүмат implementation да кулланылса да, аны бу исемлектә күрсәтү ялгыш, чөнки анда тасвирланган TVal тибы Push һәм Pop процедуралары башламнарында кулланыла һәм һәр модуль бер генә тапкыр – interface та яки implementation да күрсәтелә ала. Асылда implementation бүлегендә тасвирланган модульләрне interface бүлегендә тоташтырырга ярый. Тик бу программалауның начар стиле дип санала, чөнки ул модульләрдән мәгълүмат interface бүлеге тасвирлауларында кулланылмый.}

type

TPtr = ^TElem; {Стәк элементына күрсәткеч тибы.}

TElem = record { Стәк элементы тибы.}

Inf : TVal;

```
        Link : TPtr
      end;
var
  Top : TPtr;                {Стэк башына күрсәткеч.}
  Value : TVal;
          {finalization бүлегендә стәкне чистарту өчен үзгәрешле.}

 i : Byte;                   {Цикл оештыру өчен үзгәрешле.}
```

{interface бүлегендә инде тасвирланган процедура һәм функция башламнарын реализация бүлегендә параметрларсыз күрсәтеп була.}

```
procedure Push;
var
  P : TPtr;
begin
  New (P);
  P^.Inf := Val;
  P^.Link := Top;
  Top := P
end;

procedure Pop;
var
  P : TPtr;
begin
  if Top <> nil then
  begin
    Val := Top^.Inf;
    P   := Top;
    Top := P^.Link;
    Dispose (P)
  end
end;

function GetStatus;
var
```

```
  S, El : string;
procedure PrintEl(P : TPtr);
        {Стэк элементларын бастыруга чыгаручы, модуль өчен
        эчке булган процедура. interface бүлегендә
        экспортланмаган процедура һәм функция башламнары
        implementation бүлегендә һәрвакыт тулысынча
        (параметрлары белән) тасвирлана.}
begin
  if P <> nil then
  begin
    PrintEl (P^.Link);
    Str(P^.Inf:3, El);
    S := S + El;
  end;
end;
begin
  S := '';
  PrintEl(Top);
  GetStatus := S;
end;

initialization
              {Стэк башын беренчел кую.}
  Top := nil;
        {Стэкнең 10 элементтан торган башлангыч
        конфигурациясен төзү.}

  for i := 1 to 10 do Push(i);
finalization
        {Башка модульләрдә кулланылганнан соң нинди халәттә
        калганына карамыйча стэк алып торган хәтерне тәмамлап
        чистарту.}
   while Top <> nil  do  Pop (Value);

end.
unit UseStack;
        {UStack модулендә реализацияләнгән стэкне UseStack
        модуле куллана.}
```

interface

uses

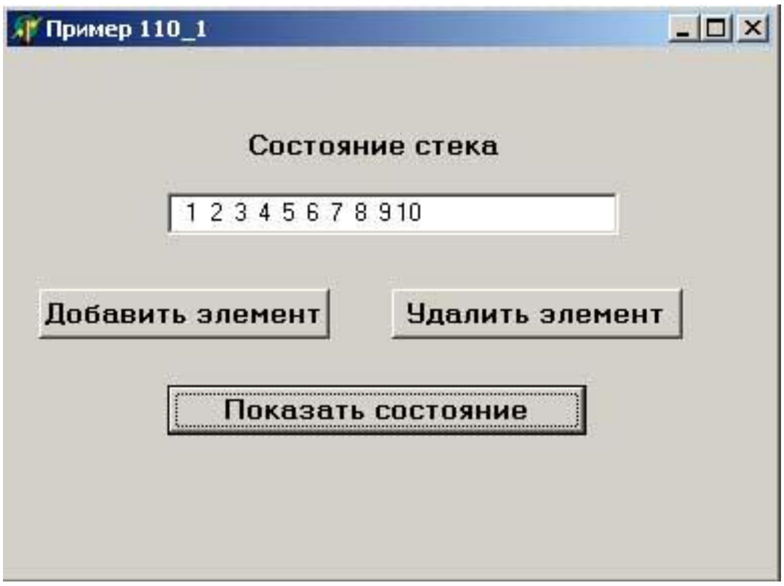
{Саналган стандарт модульләрдән алынган мәгълүмат UseStack модуле тарафыннан экспортланучы Form1 формасы һәм аның TForm1 тибын тасвирлау өчен кулланыла. Шуңа күрә Delphi компиляторы аларны implementation бүлегендә түгел, interface бүлегендә тоташтыра.}

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

```
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Label1: TLabel;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

uses
```

```
UGlob, UStack;
      { implementation бүлегендә тасвирланучы UGlob һәм
      UStack модульләрен асылда interface бүлегендә дә
      тоташтырып булыр иде. Тик бу модульләр язуның начар
      стиле булып санала, чөнки ул модульләрдән алынган
      мәгълүмат interface бүлеге тасвирламаларында
      кулланылмый. }

procedure TForm1.Button1Click(Sender: TObject);
      {Buttonl төймәсенә чирткәннән соң стәккә очраклы сан
      языла.}
var
  Val : TVal;
begin
  Val := Random(10);
  Push(Val);
end;
procedure TForm1.Button2Click(Sender: TObject);
          {Button2 төймәсенә чирткәннән соң стәкнең өске
      элементы бетерелә.}
var
  Val : TVal;
begin
  Pop(Val)
end;
procedure TForm1.Button3Click(Sender: TObject);
          {Button3 төймәсенә чирткәннән соң стәк халәте
      Edit1 кырына чыгарыла.}
begin
  Edit1.Text := GetStatus;
end;

initialization
      {Проект эшләтеп җибәрелгәндә очраклы саннар
      генераторы инициализацияләнә.}
  Randomize;
end.
```

Кушымтаның эш нәтиҗәсе 24 нче рәсемдә күрсәтелгән:



Рәс. 24. Кушымтаның эш нәтиҗәсе

2.3. Динамик библиотекалар (тупламнар)

Моңа кадәр төзелгән программалар монолит һәм чынлыкта бер башкарылучы файлдан торалар иде. Болай эшләү уңайлы, тик һәрвакытта да нәтиҗәле түгел. Әгәр дә бер программа түгел, берничә төзелсә, һәм аларның һәрберсендә дә уртак астпрограммалар җыелмасы кулланылса, бу астпрограммалар коды һәр программага кертелә. Нәтиҗәдә кодның уртак булган шактый зур күләмле өлешләре барлык программаларда да кабатлана башлый, аларны зурайта. Программалар белән эшләү катлаулана, чөнки аерым астпрограммадагы хата төзәтелсә, аны яңадан компиляцияләргә һәм кулланучыга бу астпрограммага мөрәҗәгать итүче барлык программаларны күрсәтергә кирәк була. Бу проблеманы чишү ысулы булып башкарылучы файлларны модульле оештыруга күчү тора. Delphi мохитендә бу идея динамик рәвештә йөкләнүче библиотекалар (тупламнар) ярдәмендә реализацияләнә.

2.3.1. Динамик йөкләнүче библиотекалар (тупламнар)

Динамик йөкләнүче библиотека (инглизчә dynamically loadable library) – ул астпрограммалар библиотекасы (тупламы), аны кулланучы программаның эш вакытында җәһәт хәтергә йөкләнә һәм программага тоташтырыла (компиляция һәм җыю вакытында түгел). Динамик йөкләнүче библиотекалар (тупламнар) файллары Windows мохитендә гадәттә .dll өстәмәсе белән бирелә (инглизчә Dynamic Link Library).

Кыскалык өчен динамик библиотека яки библиотека термины кулланыла, бу очракта DLL-библиотека күз алдында тотыла.

Төрле берничә программа эш вакытында бер уртак динамик йөкләнүче библиотека кулланырга мөмкин. Бу вакытта операцион система чынбарлыкта җәһәт хәтергә библиотеканың бер генә нөхсәсен (копиясен) йөкли, һәм аңа барлык программалар өчен уртак эш мөмкинлеге ача. Һәм тагын, мондый библиотекалар программаның эш барышында җәһәт хәтергә динамик йөкләнә һәм аннан алына ала, шулай итеп башка мәсьәләләр өчен системаның ресурслары бушый.

Динамик йөкләнүче библиотекаларның мөһим сыйфаты – төрле программалау телләрендә язылган астпрограммаларның үзара тәэсир итешә алуы. Мәсәлән, Delphi мохитендә башка программалау системаларында C һәм C++ телләре ярдәмендә эшләнгән динамик йөкләнүче библиотекаларны кулланып була. Кире раслау да дөрес – Delphi мохитендә төзелгән динамик йөкләнүче библиотекаларны башка программалау телләрендәге программаларга ялгап була.

2.3.2. Библиотеканы төзү

Библиотеканың структурасы

Структурасы буенча библиотеканың баштагы тексты программаның баштагы текстына охшаган, тик библиотеканың тексты program түгел, ә library ачкыч сүзеннән башлана. Мәсәлән:

library SortLib;

Башламнан соң модульләрне тоташтыру секцияләре, константаларны тасвирлау, мәгълүмат типлары, үзгәрешле типлары, процедура һәм функцияләрне тасвирлаулар бара. Процедуралар һәм функцияләр – динамик йөкләнүче библиотекада булырга тиешле төп әйбер, чөнки алар гына экспортлана ала.

Библиотека җисемендә кайбер процедуралар игълан ителсә,

procedure BubleSort(var Arr: array of Integer);

procedure QuickSort(var Arr: array of Integer);

алар автомат рәвештә тыштан чакырып була торганга әйләнә дигән сүз түгел. Моны рөхсәт итү өчен, процедура исемнәрен махсус exports секциясенә урнаштырырга кирәк, мәсәлән:

```
exports
  BubleSort,
  QuickSort;
```

exports секциясендә санап үтелгән процедуралар һәм функцияләр өтер белән аерыла, секция ахырында нокталы өтер куела. exports секцияләре берничә булырга мөмкин, алар программада ирекле рәвештә урнаша ала.

Мисалда SortLib динамик йөкләнүче библиотеканың башлангыч тексты китерелгән. Анда бөтен саннар массивын күбек принциплы аралау ярдәмендә сортлаучы бер BubleSort процедурасы бар:

```
library SortLib;
procedure BubleSort(var Arr: array of Integer);
var
  I, J, T: Integer;
begin
  for I := Low(Arr) to High(Arr) - 1 do
    for J := I + 1 to High(Arr) do
      if Arr[I] > Arr[J] then
      begin
        T := Arr[I];
        Arr[I] := Arr[J];
        Arr[J] := T;
      end;
end;
exports
  BubleSort;
begin
end.
```

Динамик йөкләнүче библиотеканың башлангыч тексты *begin ... end* операторлар блогы белән тәмамлана, анда библиотеканы эшкә әзерләүче теләсә нинди операторларны өстәп була. Бу операторлар төп программа библиотеканы йөкләгән вакытта үтәлә. Безнең гади SortLib библиотекасы эшкә әзерлек таләп итми, шуңа аның операторлар блогы буш.

Астпрограммаларны экспортлау

Әгәр дә без библиотеканың компиляцияләнгән файлының эчен карый алсак, анда һәр экспортланучы астпрограмманың үз уникаль символлы исеме белән бирелүен күрер идек. Бу исемнәр таблицага җыелган һәм астпрограммаларны эзләгәндә кулланыла – алар ярдәмендә программада язылган чакырулар командаларының библиотекадагы тиңдәш процедуралар һәм функцияләрнең

адресларына динамик бәйләү урнаштырыла. Экспорт исеме булып символларның ирекле эзлеклелеге тора ала, баш һәм юл хәрефләр аерыла.

Стандарт очракта астрограмманың экспорт исеме булып аның библиотеканың башлангыч текстындагы идентификаторы санала (зур һәм юл хәрефләр хисапка алына).Мәсәлән, exports секциясенең рәвеше түбәндәгечә булса,

exports

BubleSort;

бу процедураның экспорт исеме ‘BubleSort’ булачагын аңлата.Теләгәндә бу исемне программадагы исемнән аермалы итеп була, бу очракта тасвирлау *name* директивасы белән тулыландырыла:

exports

BubleSort name 'BubleSortIntegers';

Нәтиҗәдә BubleSort процедураның экспорт исеме ’BubleSortIntegers’ булачак.

Астпрограммаларның экспорт исемнәре библиотека чикләрендә уникаль булырга тиеш, шуңа аларны яңадан йөкләнгән *(overload)* процедуралар һәм функцияләр өчен һәрвакыт ачыктан-ачык күрсәтергә кирәк. Мәсәлән, уртак QuickSort исемле яңадан йөкләнгән ике процедура бар икән,

procedure QuickSort(var Arr: array of Integer); overload; // бөтен саннар өчен

procedure QuickSort(var Arr: array of Real); overload; // реаль саннар өчен

экспортлаганда бу ике процедурага бер-берсеннән аермалы экспорт исемнәрен ачыктан-ачык күрсәтергә кирәк:

exports

QuickSort(var Arr: array of Integer) name 'QuickSortIntegers';

QuickSort(var Arr: array of Real) name 'QuickSortReals';

Параметрларның тулы исемлекләре компилятор һәр очракта кайсы процедура турында сүз барганын ачыклый алсын өчен кирәк.

Астпрограммаларны чакыру турында килешүләр

Төрле программалау телләрендә астпрограммалар чакыруның төрле кагыйдәләре кулланыла, алар белән ярашу өчен Delphiда *register, stdcall, pascal* һәм *cdecl* директивалары бар. Бу директивалар белән эшләү башка телләрдә язылган программаларда кулланылучы динамик йөкләнүче библиотекалар төзегәндә аеруча актуаль.

Директиваларны куллануны ачыклау өчен, астпрограммаларны чакыру механизмына мөрәҗәгать итик. Ул стәк куллануга нигезләнгән.

Стәк ул – хәтер өлкәсе, аңа мәгълүмат туры тәртиптә урнаштырыла, кире тәртиптә алына, бу корал магазинына патроннар тутыру һәм бушатуга тиң. Стәктә элементлар белән эшләү тәртибе LIFO термины белән билгеләнә (инглизчәдән Last In, First Out – соңгы керде, беренче чыкты).

Искәрмә. Элементлар белән эшләүнең гадәттәге тәртибе дә бар, ул FIFO термины белән билгеләнә (инглизчәдән First In, First Out – беренче керде, беренче чыкты).

Һәр программа өчен эш вакытына үз стәге төзелә. Аның аша астпрограммалар параметрлары тапшырыла һәм анда ук бу астпрогрммаларның кайту адреслары саклана. Стәк аркасында астпрограммалар бер-берсен, хәтта рекурсив рәвештә үз-үзләрен чакыра алалар.

Астпрограмманы чакыру стәккә бөтен аргументларны һәм киләсе команданың адресын (аңа кайту өчен) кертүдән, аннары идарәне астпрограмма башына күчерүдән тора. Астпрограмманың эше беткәч, стәктән кайту адресы алына һәм идарә бу адреска күчерелә; шул ук вакытта стәктән аргументлар кысып чыгарыла. Стәкне чистарту башкарыла. Бу – эшнең гомуми схемасы, аның төрле реализацияләре була. Аерым алганда, аргументлар стәккә туры тәртиптә (сулдан уңга, астпрограмманың тасвирлавында санап үтелгәнчә), яки кире тәртиптә (уңнан сулга), яки, гомумән, стәк аша түгел, ә эш тизлеген арттыру өчен процессорның буш регистрлары аша урнаштырыла ала. Аннан тыш, стәкне чистартуны чакырылучы астпрограмма яки чакыручы программа башкара ала. Конкрет килешүне сайлауны *register, pascal, cdecl* һәм *stdcall* директивалары тәэмин итәләр. Аларның мәгънәләрен астпрограммалар чакыру турындагы килешүләр таблицасы ачыклый.

Директива	Аргументларны стәккә урнаштыру тәртибе	Стәкне чистарту өчен җаваплы	Аргументларны регистр аша тапшыру
register	Сулдан уңга	Астпрограмма	Әйе
pascal	Сулдан уңга	Астпрограмма	Юк
cdecl	Уңнан сулга	Чакыручы программа	Юк
stdcall	Уңнан сулга	Астпрограмма	Юк

Искәрмә

register директивасы барлык аргументлар да мәҗбүри процессор регистрлары аша тапшырыла дигән сүз түгел. Әгәр аргументлар саны буш регистрлар саныннан күбрәк булса, аргументларның бер өлеше стэк аша тапшырыла.

Динамик йөкләнүче библиотекаларның процедуралары һәм функцияләре өчен нинди чакыру турындагы килешүне сайларга кирәк дигән сорау туа. Җавап –*stdcall* килешүе*:*

procedure BubleSort(var Arr: array of Integer); stdcall;

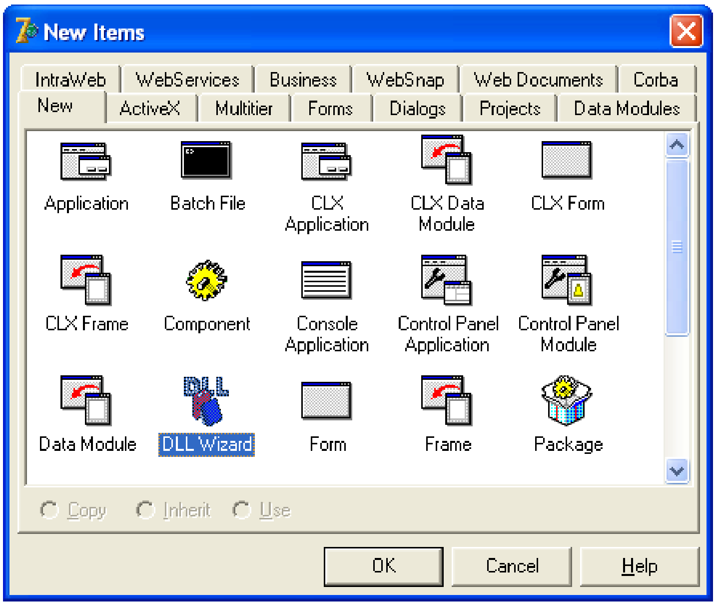
procedure QuickSort(var Arr: array of Integer); stdcall;

Башта операцион системаның астпрограммаларын чакыру өчен каралган stdcall килешүе төрле программалау телләрендә язылган программалар һәм библиотекаларның үзара тәэсир итешүе өчен туры килә. Барлык программалар да алаймы-болаймы операцион системаның функцияләрен куллана, димәк алар мәҗбүри stdcall килешүенә ярашлы булалар.

Библиотека мисалы

Бу мисалда берничә программа модуленнән төзелгән динамик йөкләнүче библиотеканың ничек рәсмиләштерелүе күрсәтелә.

1 нче адым. Delphi мохитен эшләтеп җибәрергә һәм сайлакта *File | New | Other...* командасын сайларга. Экранда ачылган диалог тәрәзәсендә DLL Wizard тамгасын сайларга һәм OK төймәсенә басарга (25 нче рәсем):



Рәс. 25. Яңа проект сайлау тәрәзәсе, DLL Wizard пункты билгеләнгән

Delphi мохите библиотека өчен кирәкле юллар белән яңа проект төзи:

```
library Project1;
{ Important note about DLL memory management ... }
uses
  SysUtils,
  Classes;
begin
end.
```

2 нче адым. *File* | *New* | *Unit* командасы ярдәмендә проектта яңа программа модуле төзәргә. Аның әзер юллары түбәндәгечә булачак:

```
unit Unit1;
interface
implementation
end.
```

3 нче адым. Модульне SortUtils.pas исеме белән сакларга, проектны SortLib.dpr исеме белән сакларга. Проектның төп файлына күчәргә һәм *uses* секциясеннән SysUtils һәм Classes модульләрен алып ташларга (алар хәзер кирәк түгел). Төп программа модуле түбәндәге рәвеш алырга тиеш:

```
library SortLib;
{ Important note about DLL memory management ... }
uses
  SortUtils in 'SortUtils.pas';
begin
end.
```

4 нче адым. SortUtils модуленең башлангыч текстын җыярга:

```
unit SortUtils;
interface
procedure BubleSort(var Arr: array of Integer); stdcall;
procedure QuickSort(var Arr: array of Integer); stdcall;
 exports
  BubleSort name 'BubleSortIntegers',
  QuickSort name 'QuickSortIntegers';
implementation
```

```
procedure BubleSort(var Arr: array of Integer);
var
  I, J, T: Integer;
begin
  for I := Low(Arr) to High(Arr) - 1 do
    for J := I + 1 to High(Arr) do
      if Arr[I] > Arr[J] then
      begin
        T := Arr[I];
        Arr[I] := Arr[J];
        Arr[J] := T;
      end;
end;

procedure QuickSortRange(var Arr: array of Integer; Low, High: Integer);
var
  L, H, M: Integer;
  T: Integer;
begin
  L := Low;
  H := High;
  M := (L + H) div 2;
  repeat
    while Arr[L] < Arr[M] do
      L := L + 1;
    while Arr[H] > Arr[M] do
      H := H - 1;
    if L <= H then
    begin
      T := Arr[L];
      Arr[L] := Arr[H];
      Arr[H] := T;
      if M = L then
        M := H
      else if M = H then
        M := L;
      L := L + 1;
```

```
    H := H - 1;
   end;
  until L > H;
  if H > Low then QuickSortRange(Arr, Low, H);
  if L < High then QuickSortRange(Arr, L, High);
end;

procedure QuickSort(var Arr: array of Integer);
begin
  if Length(Arr) > 1 then
    QuickSortRange(Arr, Low(Arr), High(Arr));
end;

end.
```

Бу модульдә BubleSort һәм QuickSort процедуралары саннар массивын ике ысул белән сортлый: «күбек» ысулы һәм «тиз» ысул кулланып. Безне хәзер процедураларны библиотекадан экспортлау өчен дөрес итеп язу кызыксындыра.

BubleSort һәм QuickSort процедураларын игълан иткәндә кулланылган *stdcall* директивасы

```
procedure BubleSort(var Arr: array of Integer); stdcall;
procedure QuickSort(var Arr: array of Integer); stdcall;
```

процедураларны Delphi да язылган программалардан гына түгел, C/C++ телләрендә язылган программалардан да чакырырга мөмкинлек бирә.

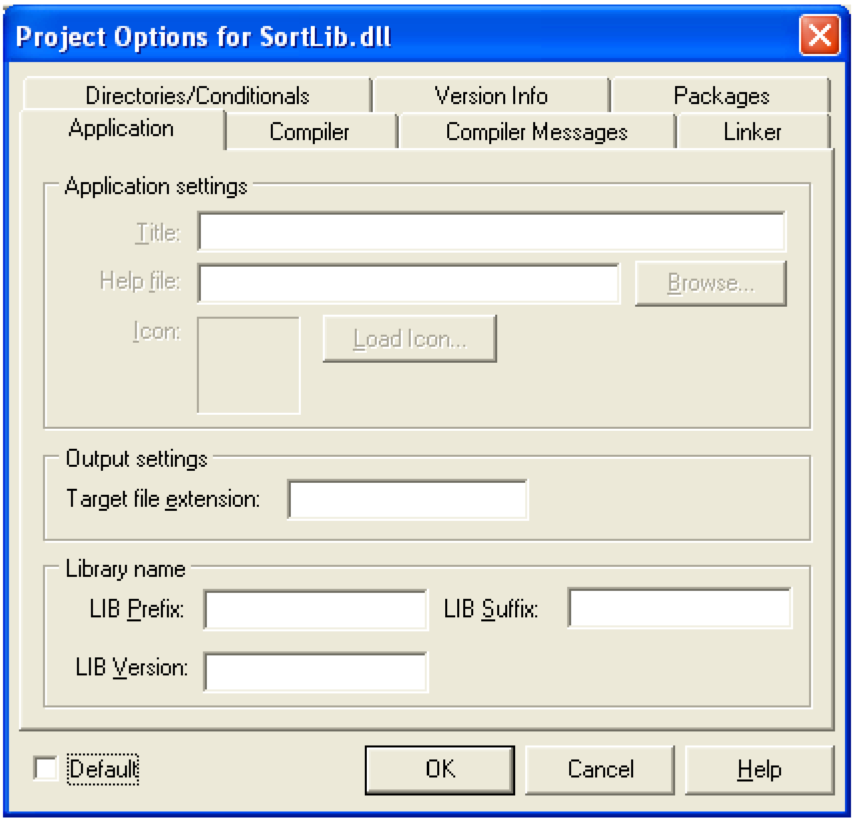
Модульдә *exports* секциясе булганга,

```
exports
  BubleSort name 'BubleSortIntegers',
  QuickSort name 'QuickSortIntegers';
```

Модульне библиотеканың төп файлында тоташтыру автомат рәвештә процедуралар экспортына китерә.

5 нче адым. Проектның барлык файлларын сакларга һәм компиляцияне башкарырга. Нәтиҗәдә дискның эш каталогында библиотеканың SortLib.dll икешәрле файлын алачаксыз. Кирәкле өстәмә файлга автомат рәвештә билгеләнә, әгәр дә компиляторның башка өстәмә билгеләве кирәк икән, сайлакның *Project | Options…* командасын кулланырга, һәм килеп чыккан Project Options тәрәзәсенең

Application кыстыргычында Target file extension кырына файл өстәмәсен язарга. (26 нчы рәсем).



Рәс. 26. Проект параметрларын көйләү тәрәзәсе

Бу тәрәзәнең LIBPrefix, LIBSuffix һәм LIBVersion кырлары ярдәмендә библиотеканы җыйганда килеп чыга торган файлның исемен билгеләү кагыйдәсен күрсәтергә мөмкин. Файл исеме түбәндәге формула буенча җыела:

<LIBPrefix> + <проект исеме> + <LIBSuffix> + ’.’ + <Target file extention> + [’.’ + <LIBVersion>]

2.3.3. Библиотеканы программада куллану

Библиотеканың процедура һәм функцияләрен гамәли программада куллану өчен импортны башкарырга кирәк. Импорт библиотеканы җәһәт хәтергә йөкләүне һәм программада язылган чакыру командаларын библиотеканың тиндәш процедура һәм функцияләрнең адресларына бәйләүне тәэмин итә. Импортның уңайлыклары һәм сыгылмалыклары белән аерыла торган ике ысулы бар:

- *статик импорт* (компиляторның external директивасы белән тәэмин ителә);

- *динамик импорт* (LoadLibrary һәм GetProcAddress функцияләре белән тәэмин ителә).

Статик импорт уңайлырак, динамик – сыгылмалырак була.

Статик импорт

Статик импортны кулланганда библиотеканы йөкләү һәм тоташтыру белән бәйле барлык гамәлләр операцион система тарафыннан автомат рәвештә төп программаны эшләтеп җибәргәндә башкарыла. Статик импортны эшкә җигү өчен программада библиотеканың процедура һәм функцияләрен тышкы итеп игълан итү җитә. Бу *external* директивасы ярдәмендә башкарыла, мәсәлән:

procedure BubleSortIntegers(var Arr: array of Integer); stdcall;
external 'SortLib.dll';
procedure QuickSortIntegers(var Arr: array of Integer); stdcall;
external 'SortLib.dll';

external ачкыч сүзеннән соң библиотеканың икешәрле файлы исеме константа юл яки константа юллы аңлатма рәвешендә языла. *external* директивасы белән инде мәгълүм *name* директивасы кулланыла ала, ул библиотекадагы процедураның экспорт исемен ачыктан-ачык күрсәтү өчен хезмәт итә. Аның ярдәмендә процедураларны игълан итүне башкача күчереп язып була:

procedure BubleSort(var Arr: array of Integer); stdcall;
external 'SortLib.dll' name 'BubleSortIntegers';
procedure QuickSort(var Arr: array of Integer); stdcall;
external 'SortLib.dll' name 'QuickSortIntegers';

Программага югарыда китерелгән барлык игъланнарны урнаштырып, BubleSort һәм QuickSort процедураларын алар программаның үз өлеше булган шикелле итеп чакырып була.

6 нчы адым. Яңа консоль программасын төзергә. Моның өчен сайлакта *File | New | Other...* командасын сайларга һәм ачылган диалог тәрәзәсендә *Console Application* тамгасын билгеләргә. Аннары OK төймәсен басарга.

7 нче адым. Программага BubleSort һәм QuickSort процедураларның external-игъланнарын өстәргә, шулай ук түбәндә китерелгән программа текстын җыярга. Проектны TestStaticImport.dpr исеме астында сакларга.

program TestStaticImport;
{$APPTYPE CONSOLE}

```
procedure BubleSort(var Arr: array of Integer); stdcall;
  external 'SortLib.dll' name 'BubleSortIntegers';
procedure QuickSort(var Arr: array of Integer); stdcall;
  external 'SortLib.dll' name 'QuickSortIntegers';
var
  Arr: array [0..9] of Integer;
  I: Integer;
begin
  // «Күбек» методы
  Randomize;
  for I := Low(Arr) to High(Arr) do
    Arr[I] := Random(100); // Массивны очраклы саннар белән тутыру
  BubleSort(Arr);
  for I := Low(Arr) to High(Arr) do
    Write(Arr[I], ' ');
  Writeln;
  // Тиз сортлау ысулы
  for I := Low(Arr) to High(Arr) do
    Arr[I] := Random(100); // Массивны очраклы саннар белән тутыру
  QuickSort(Arr);
  for I := Low(Arr) to High(Arr) do
    Write(Arr[I], ' ');
  Writeln;
  Writeln('Press Enter to exit...');
  Readln;
end.
```

8 нче адым. Компиляцияне башкарырга һәм программаны эшләтеп җибәрергә. Әгәр дә саннар экранда үсү тәртибендә бастырылса, сортлау дөрес эшли.

Үтәлгән гамәлләрдән беренче мөһим нәтиҗә чыгарырга мөмкин: программаны компиляцияләү компиляцияләнгән библиотека булуын таләп итми, бу алар бер-берсеннән бәйсез, төрле кешеләр тарафыннан төзелә ала дигән сүз. Бары тик процедура һәм функцияләргә тапшырылучы параметрлар исемлеге һәм типлар турында алдан сөйләшергә һәм бер үк чакыру килешүен сайларга кирәк.

Импорт модуле. Динамик йөкләнүче библиотекалар төзегәндә һәрвакыт аларны уңайлы куллану турында уйларга кирәк. Мәсәлән, соңгы мисалны карыйк, библиотекада бер түгел, йөз процедура, һәм алар бер программада түгел, берничәдә кирәк дип күз алдына китерик. Бу очракта процедураларның external-игъланнарын барлык программаларга *uses* секциясендә тоташтырыла торган аерым модульгә чыгару күпкә уңай. Мондый модульне шартлы рәвештә импорт модуле дип атыйлар. Тышкы астпрограммаларның игъланнарыннан тыш анда гадәттә астпрограммалар куллана торган константалар һәм мәгълүмат типлары билгеләмәләре була.

SortLib библиотекасы өчен импорт модуле түбәндәгечә күренә:

```
unit SortLib;
interface
procedure BubleSort(var Arr: array of Integer); stdcall;
procedure QuickSort(var Arr: array of Integer); stdcall;
implementation
const
  DllName = 'SortLib.dll';
procedure BubleSort(var Arr: array of Integer); external
  DllName name 'BubleSortIntegers';
procedure QuickSort(var Arr: array of Integer); external
  DllName name 'QuickSortIntegers';
end.
```

Башкарылучы файл һәрвакытта да импорт модуле белән бергә барырга тиеш, бу кулланучы астпрограмманың параметрларын ачыклый алсын һәм библиотеканы дөрес куллана алсын өчен кирәк.

Динамик импорт. Библиотеканы йөкләү һәм тоташтыру гамәлләрен (статик импорт вакытында автомат рәвештә башкарыла) операцион системаның стандарт функцияләренә мөрәҗәгать итеп мөстәкыйль башкарып була. Шул рәвешле, импортны программа эше вакытында (эшләтеп җибәргәндә түгел) динамик үтәп була.

Динамик импорт өчен библиотеканы LoadLibrary функциясен чакырып җәһәт хәтергә йөкләргә, аннары GetProcAddress функциясе ярдәмендә аннан астпрограммаларның адресларын алырга кирәк. Алынган адресларны процедур үзгәрешлеләрнең тиешле типларында сакларга кирәк. Моннан соң библиотеканың астпрограммаларын чакыру процедур үзгәрешлеләргә мөрәҗәгать итү юлы белән үтәлә

ала. Библиотека белән эшне тәмамлау өчен FreeLibrary функциясен чакырырга кирәк.

Түбәндә LoadLibrary, FreeLibrary һәм GetProcAddress функцияләренең кыскача тасвирламалары китерелгән.

- LoadLibrary (LibFileName: PChar): HModule – дискта LibFileName исемле файлда сакланучы библиотеканы җәәт хәтергә йөкли. Уңышлы үтәлгәндә функция библиотеканың санлы тасвирламасын кайтара, ул киләчәктә библиотека белән идарә итү өчен кулланылырга тиеш. Әгәр дә библиотеканы йөкләгәндә ниндидер хата килеп чыкса, нуль кыйммәте кайтарыла. LibFileName аргументында файл исеме маршрутсыз булса, бу файл түбәндәге каталогларда эзләнә: төп программа эшләтеп җибәрелгән каталогта, агымдагы каталогта, Windows операцион системасының система каталогында (аның төгәл маршрутын GetSystemDirectory функциясен чакырып белеп була), операцион система урнаштырылган каталогта (аның төгәл маршрутын GetWindowsDirectory функциясен чакырып белеп була), PATH әйләнәсе үзгәрешлесендә санап үтелгән каталогларда.
- FreeLibrary (LibModule: HModule): Bool – LibModule тасвирлагычы белән бирелгән библиотеканы җәһәт хәтердән чыгара һәм системаның библиотека алып торган ресурсларын бушата.
- GetProcAddress (Module: HModule; ProcName: PChar): Pointer – Module тасвирлагычлы библиотекадагы ProcName исемле астпрограмма адресын кайтара. Әгәр дә ProcName исемле астпрограмма библиотекада булмаса, функция nil кыйммәте (буш күрсәткеч) кайтара.

Түбәндә китерелгән TestDynamicImport программасы функциональ яктан TestStaticImport программасына тиң, тик статик импорт урынына динамик импорт техникасы куллана:

```
program TestDynamicImport;
{$APPTYPE CONSOLE}
uses
  Windows;
type
  TBubleSortProc = procedure (var Arr: array of Integer); stdcall;
  TQuickSortProc = procedure (var Arr: array of Integer); stdcall;

var
  BubleSort: TBubleSortProc; // BubleSort функциясенә күрсәткеч
```

```
  QuickSort: TQuickSortProc; //QuickSort функциясенә күрсәткеч
  LibHandle: HModule;        // библиотека тасвирлагычы

  Arr: array [0..9] of Integer;
  I: Integer;

begin
  LibHandle := LoadLibrary('SortLib.dll');
  if LibHandle <> 0 then
  begin
   @BubleSort := GetProcAddress(LibHandle, 'BubleSortIntegers');
   @QuickSort := GetProcAddress(LibHandle, 'QuickSortIntegers');
    if (@BubleSort <> nil) and (@QuickSort <> nil) then
    begin
      Randomize;
      for I := Low(Arr) to High(Arr) do
        Arr[I] := Random(100);
      BubleSort(Arr);
      for I := Low(Arr) to High(Arr) do
        Write(Arr[I], ' ');
      Writeln;
      for I := Low(Arr) to High(Arr) do
        Arr[I] := Random(100);
      QuickSort(Arr);
      for I := Low(Arr) to High(Arr) do
        Write(Arr[I], ' ');
      Writeln;
    end
    else
      Writeln('Библиотекада процедура булмау хатасы.');
    FreeLibrary(LibHandle);
  end
  else
    Writeln('Библиотеканы йөкләү хатасы.');
  Writeln('Press Enter to exit...');
  Readln;
end.
```

Программада ике процедура тибы билгеләнгән, алар параметрлар исемлеге һәм чакыру кагыйдәсе буенча (stdcall) библиотекадагы BubleSort һәм QuickSort сортлау программаларына туры килә:

```
type
  TBubleSortProc = procedure (var Arr: array of Integer); stdcall;
  TQuickSortProc = procedure (var Arr: array of Integer); stdcall;
```

Бу мәгълүмат типлары астпрограммалар адресларын саклый торган процедур үзгәрешлеләрне игълан итү өчен кирәк:

```
var
  BubleSort: TBubleSortProc;
  QuickSort: TQuickSortProc;
```

var секциясендә LoadLibrary функциясе тарафыннан кайтарыла торган, библиотеканың бөтен санлы тасвирлагычын саклау өчен кулланылучы үзгәрешле игълан ителгән:

```
var
  ...
  LibHandle: HModule;
```

Программа үз эшен LoadLibrary функциясен чакырудан башлый, аңа DLL-библиотекасы файлының исемен тапшыра.

Функция библиотеканың тасвирлагычын кайтара, ул LibHandle үзгәрешлесендә саклана.

```
  LibHandle := LoadLibrary('SortLib.dll');
  if LibHandle <> 0 then
  begin
    ...
  end
```

Тасвирлагычның кыйммәте нульдән аермалы булса, библиотека дискта табылган һәм җәһәт хәтергә уңышлы йөкләнгән дигән сүз. Моңа инангач, программа GetProcAddress функциясенә астпрограммалар адресларын алырга мөрәҗәгать итә. Алынган адреслар тиешле процедур үзгәрешлеләрдә саклана:

```
@BubleSort := GetProcAddress(LibHandle, 'BubleSortIntegers');
@QuickSort := GetProcAddress(LibHandle, 'QuickSortIntegers');
```

Һәр үзгәрешле исеме алдыннан @ символы куллануга игътибар итегез. Ул астпрограмма чакырылмавы, ә аның адресы белән эш баруын күрсәтә.

Әгәр бу адрес nil кыйммәте булмаса, күрсәтелгән исемле астпрограмма библиотекада табылган һәм аны процедур үзгәрешлегә мөрәҗәгать итеп чакырып була дигән сүз:

```
if (@BubleSort <> nil) and (@QuickSort <> nil) then
begin
  ...
  BubleSort(Arr);
  ...
  QuickSort(Arr);
  ...
end
```

Сортлау беткәч программа FreeLibrary функциясен чакырып библиотеаны чыгара.

Динамик импорт статик импорт белән чагыштырганда программалауга күбрәк көч куюны таләп итә, тик аның өстенлекләре бар:

- Җәһәт хәтер ресурсларын эффектив куллану, чөнки библиотеканы кирәк вакытта йөкләп яки чыгарып була;
- Динамик импорт библиотекада кайбер процедуралар һәм функцияләр булмаганда булыша. Статик импортта мондый очракларны операцион система эшкәртә, ул хата турында хәбәрчыгара һәм программа эшен туктата. Тик динамик импортта нәрсә эшлисе икәнен программа үзе хәл итә, һәм ул беркадәр мөмкинлекләрен өзеп куеп эшен дәвам итәргә мөмкин.

Динамик импорт җайланмалар драйверлары библиотекалары белән эшләү өчен бик әйбәт туры килә. Мәсәлән, ул Delphi мохите тарафыннан мәгълүмат базалары драйверлары белән эшләгәндә кулланыла.

2.3.4. C++ телендәге программадан библиотеканы куллану

Delphi мохитендә төзелгән библиотекаларны башка программалау телләрендә кулланырга мөмкин, мәсәлән, C++ телендә. C++ теле системалы программалау теле буларак киң кулланыш тапты, һәм программалаучыларга кайбер очракларда аңа мөрәҗәгать итәргә туры килә.

Түбәндә C++ телендә BubleSort һәм QuickSort астпрограммаларының импортын ничек башкарырга кирәк икәне күрсәтелгән.

```
extern "C" __declspec(dllimport)
void __stdcall BubleSort(int* Array, int HighIndex);

extern "C" __declspec(dllimport)
void __stdcall QuickSort(int* Array, int HighIndex);
```

Синтаксисның ваклыкларына игътибар итмичә, С++ телендә астпрограмма параметрларында ачык массивлар булмавын искәртеп китик. Аңа карамастан, программалаучы мондый астпрограммаларны чакыра ала, бу очракта ул ачык масивның ике параметрдан: массив башына күрсәткечтән һәм соңгы элемент номерыннан торуына нигезләнә.

2.3.5. Глобаль үзгәрешлеләр һәм константалар

Библиотекада игълан ителгән глобаль үзгәрешлеләр һәм константалар экспортлана алмыйлар, шуңа күрә аларга кулланучы программадан мөрәҗәгать итәргә кирәк булса, моны кыйммәт кайтаручы функцияләр ярдәмендә башкарырга кирәк.

Библиотека бер үк вакытта берничә программага тоташтырыла алса да, аның глобаль үзгәрешлеләре уртак түгел һәм программалар арасында мәгълүмат алышу өчен кулланыла алмыйлар. Библиотеканы программага тоташтырган саен операцион система яңа глобаль үзгәрешлеләр күплеге төзи, шуңа күрә библиотекага ул бер программа белән эшли булып тоела. Нәтиҗәдә программалаучылар берничә программаның бер библиотека белән эшен көйләү кирәклегеннән азат.

2.3.6. Инициализация һәм библиотека эшен тәмамлау

Библиотеканың инициализациясе аны программага тоташтырганда башкарыла һәм библиотеканың барлык модульләрендә һәм төп программа блогында initialization секциясен үтәүдән тора. Библиотеканың эшен тәмамлау библиотеканы программадан өзгәндә башкарыла; бу моментта һәр модульдә finalization секциясе үтәлә. Бу мөмкинлектән библиотека системаның ниндидер ресурсларын сораганда һәм бушатканда, мәсәлән, файллар яки мәгълүмат базалары белән тоташтыру урнаштырылганда кулланырга кирәк. Ресурсны сорау initialization секциясендә, аны бушату – finalization секциясендә башкарыла.

Алдан билгеләнгән DllProc үзгәрешлесен куллануга нигезләнгән библиотеканы инициализацияләү һәм эшен бетерүнең тагын бер

ысулы бар. DllProc үзгәрешлесе процедура адресын саклый, ул процедура библиотеканы программадан өзгәндә, шулай ук DLL-библиотеканы кулланучы программаларда параллель агымнарны юк иткәндә автомат рәвештә чакырыла. Түбәндә DllProc үзгәрешлесен куллану мисалы китерелгән:

```
library MyLib;
var
  SaveDllProc: TDLLProc;
procedure LibExit(Reason: Integer);
begin
  if Reason = DLL_PROCESS_DETACH then
  begin
    ...              // библиотека эшен бетерү
  end;
  SaveDllProc(Reason);    // алдагы процедураны чакыру
end;
begin
  ...                // библиотеканы инициализацияләү
SaveDllProc := DllProc;  // алдагы процедураны саклау
  DllProc := @LibExit;    // LibExit процедурасын билгеләү
end.
```

LibExit процедурасы бер бөтен санлы аргумент кабул итә, ул аргумент чакыру сәбәбен аныклый. Аргументның мөмкин булган кыйммәтләре:

- DLL_PROCESS_DETACH – программаны өзү;
- DLL_PROCESS_ATTACH – программаны ялгау;
- DLL_THREAD_ATTACH – параллель агым тудыру;
- DLL_THREAD_DETACH – параллель агымны тәмамлау.

DllProc үзгәрешлесен билгеләү төп программа блогында башкарыла, һәм алдагы кыйммәте «чылбыр буенча» чакыру өчен саклана.

DllProc үзгәрешлесен төзүгә библиотека параллель агымнар төзү һәм юк итүгә бәйле чара күрергә тиеш булганда гына мөрәҗәгать итү зарур. Башка очракларда инициализация һәм тәмамлауны initialization һәм finalization секцияләре ярдәмендә башкару әйбәтрәк.

2.3.7. Гадәттән тыш очраклар һәм астпрограмма башкару хаталары

Гадәттән тыш очраклар белән эшләү өчен Delphi мохите Windows операцион системасы чараларын куллана. Шуңа күрә, библиотекада берничек эшкәртелми торган гадәттән тыш очрак килеп чыкса, ул чакыручы программага тапшырыла. Программа бу гадәттән тыш очракны гадәти ысул белән – *try ... except ... end* операторлары кулланып эшкәртә ала. Бу кагыйдәләр Delphi мохитендә төзелгән программалар һәм DLL-библиотекалар өчен үтәлә. Әгәр программа башка программмалау телендә язылган булса, ул Delphi да язылган библиотекадагы гадәттән тыш очракны операцион системаның $0EEDFACE кодлы гадәттән тыш очрагы итеп эшкәртергә тиеш. Гадәттән тыш очрак чыгарган инструкциянең адресы ExceptionInformation массивының беренче элементында, гадәттән тыш очракны тасвирлаучы объект икенче элементында була, бу массив гадәттән тыш очрак турындагы система язмасының бер өлеше булып тора.

Әгәр дә библиотека SysUtils модулен тоташтырмаса, гадәттән тыш очракларны эшкәртеп булмый. Бу очракта библиотекада теләсә нинди хата килеп чыкканда чакыручы программаның эше туктатыла, өстәвенә программа хәтердән алып ташлана һәм аны тәмамлау коды үтәлми. Бу өстәмә хаталар чыгуга китерергә мөмкин, шуңа библиотекага SysUtils модулен тоташтырмау карары кабул ителсә, гадәттән тыш очракларны библиотеканың астпрограммаларыннан «чыгарып җибәрмәү» чарасын күрергә кирәк.

2.3.8. Хәтернең уртак идарә программасы (менеджер)

Әгәр динамик хәтерне аерып алу һәм азат итү библиотека һәм программа арасында анык яки аныксыз бүленгән булса, библиотекада да, программада да ShareMem модулен тоташтыру кирәк. Аны библиотекада да, аны кулланучы программада да uses секциясендә беренче итеп күрсәтергә кирәк.

ShareMem модуле программа белән бергә таралырга тиеш булган динамик йөкләнүче Borlndmm.dll библиотекасының импорт модуле булып тора. Инициализация вакытында ShareMem модуле хәтернең стандарт идарә программасын Borlndmm.dll библиотекасының идарә программасына алмаштыра. Шуңа күрә библиотека һәм программа хәтерне бергә аерып ала һәм азат итә алалар.

ShareMem модулен библиотека һәм программа арасында озын юллар яки динамик массивлар тапшырулар барса да тоташтырырга кирәк. Озын юллар һәм динамик массивлар динамик хәтердә урнаштырылганга һәм автомат рәвештә идарә ителгәнгә (сылтамалар санын билгеләү юлы белән), программа тарафыннан аларга бүленеп бирелүче хәтер блоклары библиотека тарафыннан азат ителә ала (һәм киресенчә). Borlndmm.dll библиотекасыннан хәтернең уртак идарә программасын куллану программаны һәм библиотеканы хәтерне яшерен җимерүләрдән саклый.

Искәрмә. Соңгы кагыйдә SortLib.dll библиотекасы төзегәндә BubleSort һәм QuickSort астпрограммаларында кулланылган ачык массив-параметрларга кагылмый.

Стандарт система үзгәрешлеләре

Delphi да System станларт модуле бар, ул һәр программага яки библиотекага аныксыз тоташтырыла. Бу модульдә алдан билгеләнгән систем астпрограммалар һәм үзгәрешлеләр бар. Алар арасында Boolean типлы IsLibrary үзгәрешлесе бар, аның кыйммәте библиотека өчен True, гадәти программа өчен False. IsLibrary үзгәрешлесенең кыйммәтен тикшереп, астпрограмма аның библиотека өлеше булу-булмавын билгели ала.

System модулендә CmdLine: PChar үзгәрешлесе дә игълан ителгән, анда программаны эшләтеп җибәргән команда юлы бар. Библиотекалар мөстәкыйль эшләтеп җибәрелә алмый, шуңа алар өчен CmdLine үзгәрешлесе һәрвакыт nil кыйммәте ала.

3. ОБЪЕКТКА ОРИЕНТЛАШКАН ПРОГРАММАЛАУНЫҢ (ООП) ТӨП ПРИНЦИПЛАРЫ

ООПның үзәк идеясы булып «абстракция» төшенчәсен гамәлгә ашыру тора. Абстракциянең мәгънәсе шунда: теләсә нинди катлаулы затны, төшенчәне эчке төзелеше һәм эше детальләренә игътибар итмичә, бербөтен итеп карап, гамәлләрне дә аның өстендә шул рәвешле эшләп була.

Программ комплекс төзегәндә кирәкле абстракцияләрне билгеләү мөһим.

Мисал: Дәресләр расписаниесе төзү.

Кирәкле **абстракцияләр**: студент, лекцияләр курсы, укытучы, аудитория.

Операцияләр:

- Студентны группага билгеләү
- Группага аудитория билгеләү

Абстракцияләр билгеләүнең төп ысулларының берсе – иерархияле классификация концепциясен куллану. Аның асылы булып катлаулы системаларның гадиерәк фрагментларга (кисәкләргә) бүленүе тора.

Гомумән алганда барлык катлаулы системалар да иерархияле, аларның иерархия дәрәҗәсе абстракцияләрнең төрле дәрәҗәсен (биеклеген, күрсәткечен) чагылдыра. Һәр конкрет мәсьәләнең үз дәрәҗәсе карала. Абстракциянең иң түбән дәрәҗәсен сайлау шактый ирекле. Бер очракта иң түбән дип сайланган дәрәҗә башка проектта хәйран югары абстракция дәрәҗәсе булырга мөмкин.

Тип иерархиясе һәм структура иерархиясен аералар, без аларны класслар структурасы һәм объектлар структурасы дип атарбыз.

Барлык объектка ориентлашкан программалау телләрендә түбәндәге төп механизмнар (постулатлар) реализацияләнгән, (гамәлгә ашырылган):

- Инкапсуляция
- Мирас итеп алу
- Полиморфизм

Бу механизмнар абстракцияләрне билгеләү һәм куллану өчен мөһим.

1) **Инкапсуляция** – кодны һәм код тәэсир итә торган мәгълүматны бәйләүче, шул ук вакытта аларны бу код өчен тышкы (чит) булган код тарафыннан кирәк булмаган тәэсир итүдән саклаучы механизм. Код һәм мәгълүмат белән эш мөмкинлеге интерфейс тарафыннан катгый саклана, контрольләнә.

ООП кулланганда инкапсуляция нигезе булып класс тора.

Инкапсуляция механизмы классның кайбер детальләрен кулланучыдан яшерергә (капсула эченә куярга) мөмкинлек бирә, бу гамәл класс объектлары белән эшләүне җиңеләйтә.

2) **Мирас итеп алу** – бу механизм ярдәмендә бер объект (төзелгән класснык, онык класснык) икенче объектның (баба класснык, база классыныкын) сыйфатларын үзенә ала. Мирас итеп алуны кулланганда яңа объектны өр яңадан тасвирлау мәҗбүри түгел, шуңа программлаучының эше күпкә җиңеләя. Мирас итеп алу объектка үзенең баба объекты (баба, база классыннан) атрибутларын

(үзлекләрен, сыйфатларын) алырга, ә үзе өчен үзенә генә хас булган, уникаль характеристикаларны билгеләргә мөмкинлек бирә.

Мирас итеп алу иерархияле классификация концепциясенә туры килүче бик мөһим төшенчә.

3) **Полиморфизм** – бер үк интерфейсны гамәлләрнең гомуми классына куллану мөмкинчелеге бирә торган механизм.

Мисал: Саклау өчен 3 төрле стэк бар:

- бөтен саннар өчен
- йөзмә нокталы саннар өчен
- символлар өчен

Объектка ориентлашкан программада өч идарә итүче астпрограмма урынына бер генә астпрограмма (бер интерфейс) кирәк булачак.

Полиморфизмның гомуми концепциясе: бер интерфейс – күп метод, ысул.

Конкрет очракка карата кирәкле ысулны (методны) сайлау компиляторга йөкләнә. Программалаучыга берничә урынына бер интерфейсны истә калдыру һәм куллану җитә, бу шулай ук эшне җиңеләйтә.

Статик (компиляция этабында функция һәм операцияләрне яңадан йөкләү ярдәмендә башкарыла), **динамик** (программа үтәлгән вакытта виртуаль функцияләр механизмы ярдәмендә башкарыла) һәм **параметрик** (компиляция этабында шаблоннар механизмын кулланып башкарыла) полиморфизм була.

Искәрмә. Каралган абстракция, инкапсуляция, мирас итеп алу, полиморфизм төшенчәләре ООП парадигмасына гына хас түгел. Бөтен саннар һәм йөзмә нокталы саннар өстендә арифметик гамәлләр процессорда төрле алгоритмнар буенча башкарыла. Бу очракта полиморфизм анык түгел, яшерен күренә.

Хәзер каралган һәм башка төшенчәләрне Delphi мохите өчен Object Pascal теленә карата өйрәник.

3.1. Класс билгеләмәсе

Класслар дип Object Pascal дә кырлары, методлары (ысуллары) һәм үзлекләре булган махсус типлар атала. Теләсә нинди башка тип кебек үк, класс реализациянең (гамәлгә ашыруның) объект дип аталучы конкрет (анык) нөхсәләре төзү өчен үрнәк булып кына тора.

Аныклап китик, Object Pascal дән элгәр Turbo Pascal дә объектлар дип Object Pascal класслары белән күп уртак сыйфатлары булган типлар атала. Тик Object Pascal нең объект моделенә кертелгән җитди камилләштерүләр телне төзүчеләрне объектларны билгеләү өчен махсус «класс» термины кертергә мәҗбүр итәләр. Сүз уңаенда, бу термин Си++ тән алынган. Элегрәк Turbo Pascal with Objects 7.0 системасында эшләнгән программалар белән ярашу өчен Object Pascal дә object тип-объекты сакланган, ул «иске» объектлы модельгә туры килә. Бу модельнең барлык мөмкинчелекләре классларга хас булганга, без аларны аерым карамабыз, «бушаган» объект терминын классның конкрет нөхсәсе реализациясен билгеләү өчен кулланачакбыз.

Классның башка типлардан мөһим аермасы: класс объектлары һәрвакыт бергә бирелә. Шуңа күрә объект-үзгәрешле асылда хәтернең динамик өлкәсенә күрсәткеч кенә булып тора.Тик башка күрсәткечләрдән аермалы буларак, объект билгеләгән әйберләргә сылтамада объект исеменнән соң «^» символын куллану тыела:

type

TMyClass = class(TObject) Field: Integer;

end;

var

MyClass: TMyClass;

begin

MyClass^.Field := 0; // Хата! Болай язарга кирәк:

MyClass.Field := 0;

end;

Класслар – программалаучылар катлаулы программалар төзүне гадиләштерү һәм сыйфатларын яхшырту өчен уйлап тапкан үзенчәлекле «әйбер». Югарыда әйтелгәнчә, класслар нигезендә инкапсуляция, мирас итеп алу һәм полиморфизм дип аталган өч нигез принцибы ята. Object Pascal дә класс дип составында үзгәрешлеләр, функцияләр һәм процедуралар булган тел структурасы атала. Үзгәрешлеләр билгеләнешләренә карап кырлар яки үзлекләр дип атала. Классның процедуралары һәм функцияләре – методлар, ысуллар. Ысуллар – класс эчендә бирелгән, тасвирланган, кырлар белән операцияләр башкарырга каралган процедуралар һәм функцияләр**.** Класс составында ысулларны чакыру өчен кирәк булган мәгълүмат урнашкан махсус таблицага күрсәткеч бар. Ысуллар

чакырылганда аларга, гадәти процедуралар һәм функцияләрдән аермалы буларак, ысулны чакырган объектка күрсәткеч тапшырыла. Шуңа күрә нәкъ ысулны чакырган объектның кырлары эшкәртелә. Ысул эчендә ысулны чакырган объектка күрсәткеч махсус self исеме астында була.

Объектка ориентлашкан программалауда кырлар дип күренү өлкәсе public булган элементларны атау каралган. Элементлар шулай ук атрибут дип атала. Ысулларны операцияләр дип тә атыйлар.

3.2. Инкапсуляция

Класс өч төшенчә – кырлар, ысуллар һәм үзлекләр берлеген гәүдәләндерә. Бу затларны бер бөтенгә берләштерү инкапсуляция дип атала. Инкапсуляция классны программаның башка өлешләреннән аерырга мөмкинлек бирә, аны конкрет мәсьәлә чишү ихтыяҗларын канәгатьләндерерлек итә. Нәтиҗә буларак, класс һәрвакыт үзендә бертөрле функциональлек йөртә. Мәсәлән, tForm классы үз составында Windows-тәрәзәләр төзү өчен кирәкле бар әйберне йөртә (инкапсуляцияли). TMemo классы – тулы функцияле текст редакторы, TTimer классы таймер белән эшне тәэмин итә, һ.б.

Объектка ориентлашкан программалауның классик кагыйдәсе түбәндәгене раслый: ышанычлылыкны тәэмин итү өчен объект кырларына турыдан туры керү, мөрәҗәгать итү кирәкми, аларны уку һәм яңарту кирәкле ысулларны чакыру ярдәмендә башкарылырга тиеш.

Бер типта кырларны һәм ул кырлар белән эшләү өчен ысулларны (функцияләрне) берләштерү инкапсуляция булачак.

Моннан тыш, инкапсуляция кырларга мөрәҗәгать итү турыдан-туры түгел (язмалар очрагында кебек), ә ысуллар ярдәмендә башкарылуын таләп итә.

Инкапсуляция – әзер программа өлешләре белән алмашу мөмкинчелеге бирә торган көчле чара. Delphi класслары тупламы – ул асылда Borland программалаучылары төзегән, сезнең программаларда куллану өчен әзер «кирпечләр» җыелмасы.

3.3. Мирас итеп алу

Теләсә кайсы класс башка класстан төзелергә мөмкин. Моның өчен аны тасвирлаганда баба классның исеме күрсәтелә:

TChildClass = class (TParentClass)

Төзелгән класс автомат рәвештә үзенең баба классының кырларын, ысулларын, үзлекләрен мирас итеп ала һәм яңалары белән тулыландыра ала. Шуңа күрә мирас итеп алу принцибы катлаулы классларны бер-бер артлы төзүне һәм яңа класс тупламалары эшләүне тәэмин итә.

Object Pascal нең барлык класслары бердәнбер TObject баба классыннан, нәсел башыннан төзелгән. Бу классның кырлары һәм үзлекләре юк, ләкин аның барлык объектларның төзүдән башлап бетерүгә кадәр яшәү циклын тәэмин итүче гомуми ысуллары бар. Программалаучы TObject классының онык классы булмаган класс төзи алмый. Түбәндәге ике белдерү бердәй:

TaClass = class(TObject) TaClass = class

Мисал. Программада ике класс төзелә. TStudent классы TPerson классының дәвамчысы, варисы, мирас итеп алучысы. TStudent классында curs дигән яңа атрибут бар.

```
program Project2;
{$APPTYPE CONSOLE}
Uses SysUtils;
Type
TPerson=class
name:String;
fam:String;
constructor Create(NewName:String;NewFam:String);
Destructor Destroy;
function GetName:String; function
GetFam:String;
end;
TStudent=class(TPerson)
curs:Integer;
constructor
Create(NewName:String;NewFam:String;NewCurs:Integer);
function GetCurs:Integer; end;
constructor
 TPerson.Create (NewName:String;NewFam:String);
 begin
name:=NewName;
fam:=NewFam;
end;
```

```
        function   TPerson.GetName:String;
begin
        GetName:=name;
        end;
        function     TPerson.GetFam:String;
begin
        GetFam:=fam;
        end;
        Destructor TPerson.Destroy ;
        begin
        end;
        constructor
      TStudent.Create(NewName:String;NewFam:String;NewCurs:Integer);
        begin
        name:=NewName;
        fam:=NewFam;
        curs:=NewCurs;
        end;
        function TStudent.GetCurs:Integer;
        begin
        GetCurs:=curs;
        end;
        Var L:TPerson; S:String; Lena:TStudent;
        begin
        L:=TPerson.Create('Leonard','Euler');
        Lena:=TStudent.Create('Elena','Petrova',3);
        S:=L.GetName;
        Writeln(S,' ',L.GetFam);
        Writeln(Lena.GetName, ' ', Lena.GetFam);
        Readln;L.Destroy;
         end.
```

1 нче кисәтү. Object Pascal дә класс нөсхәләре бары тик динамик кына була ала. Бу L үзгәрешлесе объект адресын үз эченә алучы күрсәткеч булып тора дигән сүз. Object Pascal дә классның берничә конструкторы була ала. Гадәттә конструкторга Create() исеме бирелә (Turbo Pascal дә конструкторның исеме Init(),C++тә конструктор исеме класс исеме белән бер). Детсрукторның гадәттәге исеме Destroy().

Объект аны инициаллаштыручы махсус ысул – конструкторны чакыру нәтиҗәсендә төзелә.

L:=TPerson.Create('Leonard','Euler');

Төзелгән нөхсәне башка ысул – деструктор ярдәмендә бетереп була: *L. Destroy;*

2 нче кисәтү. Инкапсуляция кырларга мөрәҗәгать итү турыдан-туры түгел (язмалар очрагында кебек), ә ысуллар ярдәмендә башкарылуын таләп итә.

S:=L.GetName;

дип язу урынына

S:=L.Name;

дип язып, ягъни кырга турыдан-туры мөрәҗәгать итеп була. Без инкапсуляция шартларына игътибар итмәдек, гәрчә бу очракта хата юк.

3 нче кисәтү. Оныk классының конструкторын түбәндәгечә күчереп язып була:

Constructor
TStudent.Create(NewName:String;NewFam:String;NewCurs:Integer);
begin
inherited Create(NewName,NewFam);
curs:=NewCurs;
end;

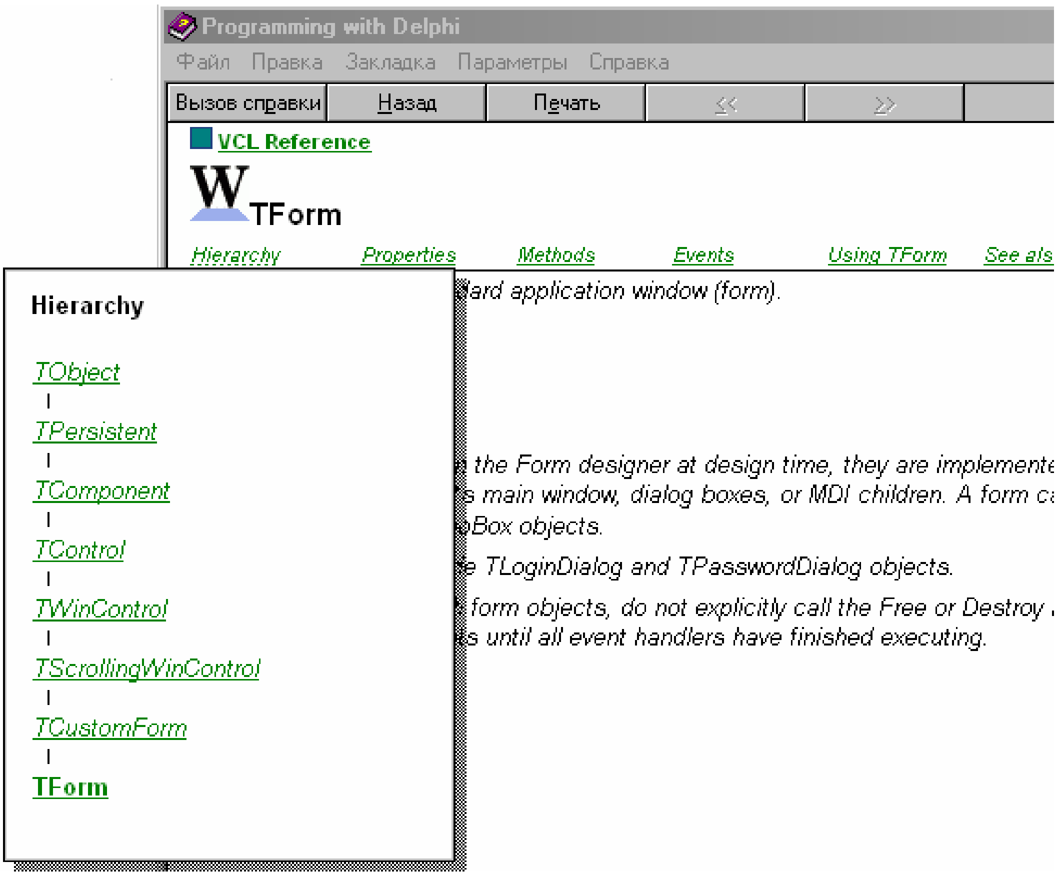
3.4. Класслар диаграммасы

Мирас итеп алу принцибы классларның тармакланган агачын төзүгә китерә. Ул нәсел башы TObject тан токымнарына күчкәндә эзлекле рәвештә үсә. Һәр токым үзенең бабасының мөмкинлекләрен яңалары белән тулыландыра һәм үзенең токымнарына тапшыра. Мисал өчен рәсемдә Delphi класслары агачының бер өлеше күрсәтелгән (27 нче рәсем). Tpersistent классы үзенең бабасы TObject ның мөмкинлекләрен баета: ул мәгълүматны файлда саклый һәм файлдан ала «белә», нәтиҗәдә аның токымнары, оныклары да бу операцияләрне эшли ала. Үз чиратында TComponent классы төзүче мохите белән үзара тәэсир итешә ала һәм бу «сәләтен» токымнарына, оныкларына тапшыра. TControl файллар һәм төзүче мохите белән генә эшләп калмый, ул экранда күренә торган сурәтләрне дә булдыра һәм аларга хезмәт күрсәтә ала, ә аның токымы TWinControl Windows-тәрәзәләр төзи ала һ.б.

Онык класста баба класстан мирас итеп алган кырлар һәм ысуллар белән эш мөмкин; әгәр ысул исемнәре тәңгәл килсә, алар бер-берсен каплый дип әйтәләр.

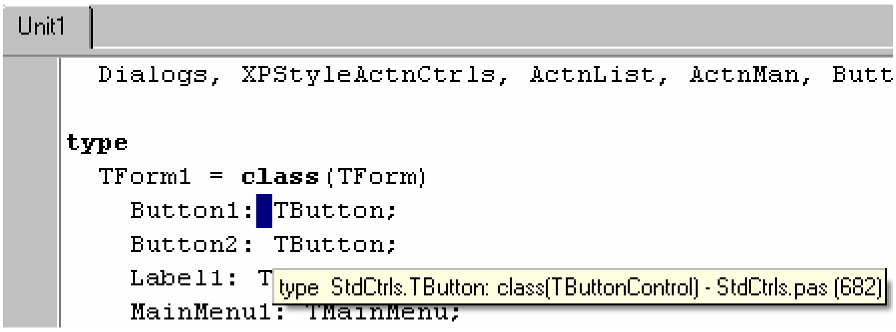
Чакырганда нинди гамәлләр үтәлүенә карап, ысуллар өч төркемгә бүленә. Беренче төркемгә статик ысуллар, икенчегә – виртуаль һәм динамик, өченчегә – Delphi 4 тән башлап кертелгән яңадан йөкләнә ала торган (overload) ысуллар керә.

Беренче төркем ысуллары онык классларда яңадан билгеләнгәндә тулысынча капланалар. Бу очракта ысулны игълан итүне тулысынча үзгәртеп була. Икенче төркем ысуллары мирас итеп алынганда атамаларын һәм типларын сакларга тиеш. Яңадан йөкләнә торган ысуллар мирас итеп алу механизмына куллану шартларына карап ысулның кирәкле вариантын (үзенеке яки бабаныкы) сайлау мөмкинлеген өстиләр.

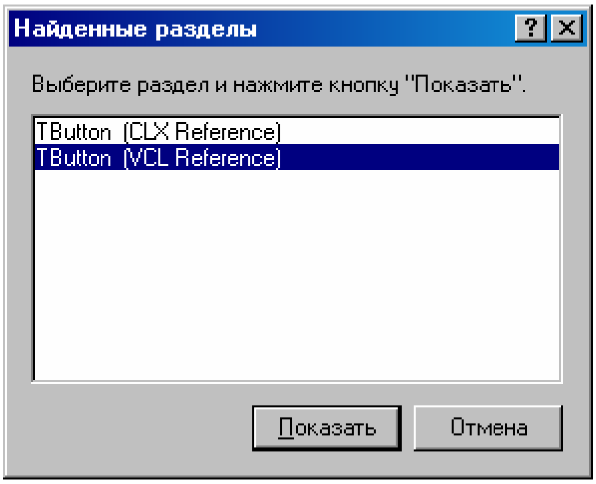


Рәс. 27. Мирас итеп алу диаграммасы мисалы. Delphi белешмә хезмәтеннән

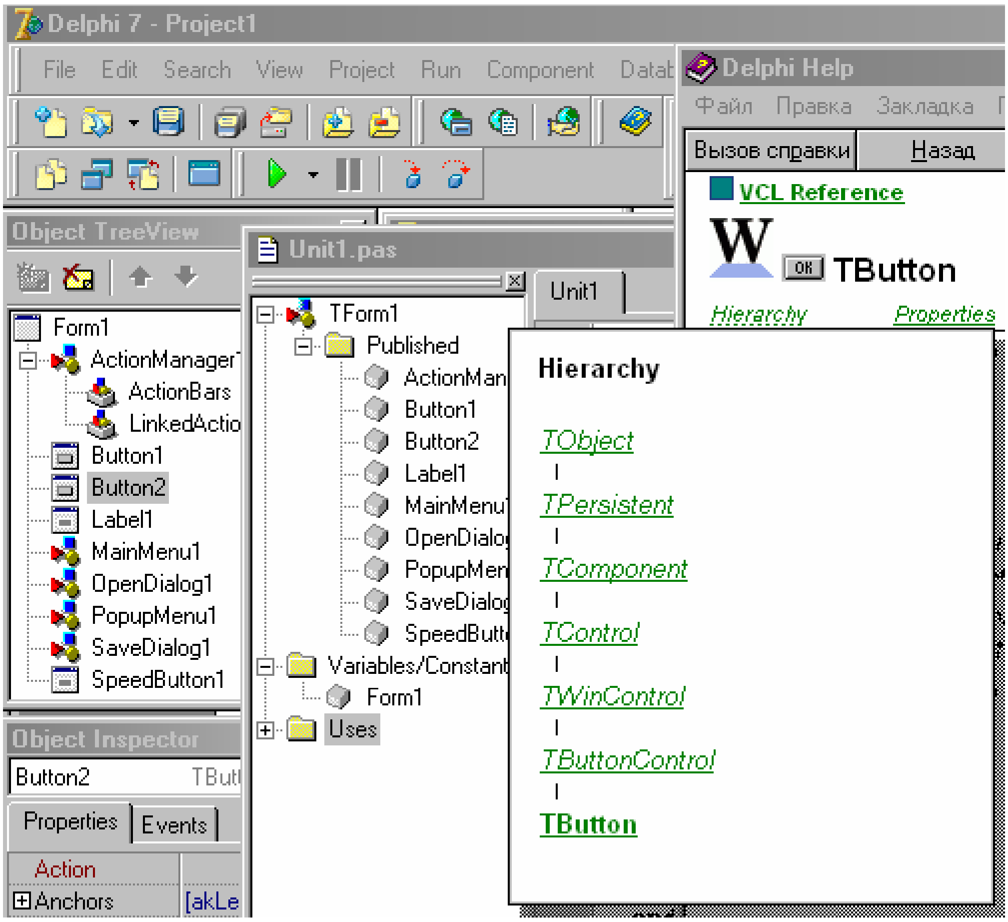
Класс турында белешмә, мәгълүмат алу өчен курсәрне класс исеменә куярга кирәк.



<F1> төймәсенә басарга



VCL Reference ны сайларга һәм «Күрсәтергә» төймәсенә чиртергә



3.5. Классны игълан итү

Теләсә нинди яңа төзелә торган классның махсус сүзләр: published (басылган, дөнья күргән), private (ябык), protected (сакланган), public (ачык) һәм automated (автоматлаштырылган) белән билгеләнгән бүлекләре булырга мөмкин. Һәр бүлекнең эчендә башта кырларны, аннары ысуллар һәм үзлекләрне билгелиләр.

Бүлекләр класс тасвирлау элементларының күренүчәнлек өлкәләрен билгелиләр. Public бүлеге анда санап үтелгән кырлар, ысуллар һәм үзлекләрнең күренүчәнлек өлкәсенә чикләүләр куймый, аларны программаның теләсә нинди башка модулендә чакырып була. Published бүлеге шулай ук күренүчәнлек өлкәсенә чикләү куймый, тик анда башкару этабында гына түгел, төзү этабында да (ягъни Объект инспекторы тәрәзәсендә) мөрәҗәгать итеп була торган үзлекләр санап үтелә. Published бүлеге стандарт булмаган компонентлар эшләгәндә генә кулланыла. Delphi мохите формага куелган компонентларның тасвирламаларын класс бүлеге башламасыннан соң урнашкан һәм беренче игълан ителгән бүлеккә кадәр дәвам иткән исемсез махсус бүлеккә урнаштыра. Бу – published бүлеге. Программалаучыга ул бүлеккә үзенең класс тасвирлау элементларын урнаштырырга яки ул бүлектән мохит тарафыннан куелган элементларны алып атарга кирәкми. Private бүлеге күренүчәнлек өлкәсен минимумга кадәр кыса: тасвирлауның ябык элементларына бирелгән класс ысуллары эчендә һәм класс тасвирланган модульдәге астпрограммаларда гына мөрәҗәгать итеп, кереп була. Private бүлегендә игълан ителгән элемент башка модульдә урнашкан классның иң якын токымнары өчен дә мөрәҗәгать итеп, кереп булмый торганга әйләнә. Protected бүлеге классның үз ысулларына һәм кайсы модульдә урнашканга бәйсез рәвештә барлык токымнарына ачык. Ниһаять, automated бүлеге автоматлаштыруның OLE-объектлар интерфейсына өстәләчәк үзлекләр һәм ысулларны игълан итү өчен генә кулланыла; бу бүлек әгъзаларының күренүчәнлек өлкәсе чикләнмәгән.

Object Pascal дә теләсә кайсы бүлекне күпме кирәк, шуның кадәр тапкыр игълан итеп була, бүлекләрнең бер-бер артлы урнашу тәртибе мөһим түгел. Теләсә кайсы бүлек буш була ала.

Түбәндәге код фрагменты күренүчәнлек өлкәләрен аңлата.

Unit Unit1;

Interface

```
Uses Controls, Forms;
type
Tform1 = class(Tform)
Button1: Tbutton;   // Бу бүлекне Delphi эшли
                    { Аның элементлары һәркемгә ачык }
private      // Бу бүлек Unit1 модулендә ачык
FintField: Integers
Procedure SetValue(Value: Integer);
Function GetValue: Integer;
published    // Бу бүлек теләсә кайсы модульдә ачык
Property IntField: read GetValue write SetValue;
protected // Бу бүлек онык классларга ачык
Procedure Proc1;
public // Бу бүлек теләсә кайсы модульдә ачык
Procedure Proc2;
end;
var Form1: Tform1;
Implementation Procedure Tform1.Proc1 ;
Button1.Color := clBtnFace;
// Болай ярый
FintField := 0;
// Болай ярый
IntField := 0;
// Болай ярый Proc1;
// Болай ярый Proc2;
// Болай ярый
end;
begin
Form1.Button1.Color := clBtnFace; // Болай ярый
Form1.FintField := 0; // Болай ярый
Form1.IntField := 0; // Болай ярый
Form1.Proc1; // Болай ярамый!
Form1.Proc2; // Болай ярый
end.
Unit Unit2;
Interface
Uses Controls, Unit1;
type
```

```
Tform2 = class(Tform1) Button2: Tbutton;
Procedure Button2Click(Sender: Tobject);
end;
var Form2: Tform2;
Implementation
Procedure Tform2.Button2Click(Sender: Tobject);
begin
Button1.Color := clBtnFace; // Болай ярый
FintField := 0; // Болай ярамый!
IntField := 0; // Болай ярый
Proc1; // Болай ярый
Proc2; // Болай ярый
end;
begin
Form1.Button1.Color := clBtnFace; // Болай ярый
Form1.FintField := 0; // Болай ярамый!
Form1.IntField := 0; // Болай ярый
Form1.Proc1; // Болай ярамый!
Form1.Proc2; // Болай ярый
end.
```

Онык классны игълан иткәндә класс элементларын бер күренүчәнлек өлкәсеннән икенчесенә күчерергә рөхсәт ителә. Югарыдагы мисал өчен мондый игълан итү мөмкин:

```
type
Tform2 = class(Tform1)
Public
Procedure Proc1;
end;
```

Моннан соң unit2 модулендә мондый мөрәҗәгать мөмкин:

```
Form2.Proc1;
```

Private бүлегенә күчерелгәннән соң тасвирлау элементы токымнарга күренми (әгәр токым, күп очрактагыча, башка модульдә игълан ителсә). Шуңа аны башка бүлеккә күчереп булмый.

Класс бары тик модульнең интерфейс өлкәсендә яки башкару бүлеге башында ук игълан ителә ала. Классны астпрограммалар тасвирлау бүлегендә билгеләргә ярамый.

Искәрмә. Өч күренүчәнлек өлкәсе – private, protected, public – ысуллар күренүчәнлеге үсү тәртибендә урнашкан. Токым классларда

ысуллар һәм үзлекләрнең күренүчәнлеген арттырып була, киметеп булмый. Онык классны тасвирлаганда ысуллар һәм үзлекләрне бер күренүчәнлек өлкәсеннән икенчесенә күчерү өчен аларны күчереп язу, тасвирлау кирәк түгел, башка урында искә алу җитә.

3.6. Виртуаль ысуллар һәм полиморфизм

Югарыда әйтелгәнчә, Delphi ның хәзерге вакытта кулланылучы версияләрендәге «яңа» объект моделендә һәр класс күрсәткеч тасвирламасы булып тора һәм барлык объектлар да динамик. Объектларны хәтердә урнаштыру белән класс конструкторы шөгыльләнә. Баба конструкторны чакыру өчен inherited инструкциясе кулланыла.

3.6.1. Статик һәм виртуаль ысуллар

Ысуллар статик һәм виртуаль була ала. Тышкы аерма шунда: виртуаль ысулның тасвирламасы башламыннан соң virtual ачкыч сүзе күрсәтелә, мәсәлән,

Procedure PushPet;virtual;

Виртуаль ысулның статик ысулдан аермасы: объект тибы нөхсәсе белән статик ысул арасында бәйләнеш компиляция вакытында урнаша, нөхсә белән виртуаль ысул арасында бәйләнеш программа башкарылу вакытында урнаша.

Статик ысулны чакыруны компиляция вакытында рөхсәт итү процессы иртә бәйләү дип атала. Соң бәйләү виртуаль ысуллар өчен кулланыла.

Әгәр баба класста ысул виртуаль дип игълан ителсә, теләсә кайсы онык класстагы шул ук исемдәге ысуллар виртуаль була. Һәр классның виртуаль ысуллар таблицасы бар (таблица виртуальных методов – ВМТ яки VMT). Бу таблицада бирелгән типка (класска) бүленгән үлчәм һәм һәр виртуаль ысул өчен бу ысулны реализацияләүче код күрсәткече бар. Конструктор нөхсә (объект) белән виртуаль ысуллар таблицасы арасында бәйләнеш урнаштыра. Һәр класс өчен виртуаль ысуллар таблицасы бер генә.

1 нче мисал:

```
program Pets2;
{$APPTYPE CONSOLE}
Type
TPet=class
```

```
Name:String;
constructor Create(NameI:String);
procedure Speak;virtual;
procedure PushPet;virtual;
end;
Type
TDog=class(TPet)
procedure Speak;override;
end;
Type
TCat=class(TPet)
procedure Speak;override;
end;
Type
TBird=class(TPet)
procedure Speak;override;
end;
constructor TPet.Create(NameI:String);
begin
Name := NameI;
end;
procedure TPet.Speak;
begin
writeln('!!!!!!!');
end;
procedure TDog.Speak;
begin
writeln('Gav!');
end;
procedure TCat.Speak;
begin
writeln('May!');
end;
procedure TBird.Speak;
begin
writeln('Kar!!');
end;
procedure TPet.PushPet;
```

```
begin
{ .... .... .... }
writeln('Pet:',Name);
Speak;
end;
Var
Bob : TDog; Boss : TCat; Vorona : TBird;
Begin
Bob:= TDog.Create('Bobik');
Boss:= TCat.Create('Bossik');
Vorona:= TBird.Create('Karkusha');
Boss.PushPet;
Vorona.PushPet;
readln;
End.
```

Искәрмә. Яңа модельдә ысулларны яңабаштан билгеләү override директивасы ярдәмендә бирелә (virtual түгел).

Исемнәре бер виртуаль ысуллар туган классларда төрле гамәлләр башкара. ООП да мондый мөмкинлек полиморфизм дип атала.

Полиморфизм – ул классларның мәгънәләре охшаш проблемаларны төрле ысуллар белән чишү үзлеге. Object Pascal дә классның үз-үзен тотышы үзлекләре аңа керүче ысуллар белән билгеләнә. Токым класста теге яки бу ысулның алгоритмын үзгәртеп, программалаучы бу токымнарга баба класста булмаган аерым үзлекләр бирергә мөмкин. Ысулны үзгәртү өчен токымда аны капларга, ягъни токымда шул ук исемдәге ысулны игълан итәргә һәм анда зарур гамәлләрне башкарырга кирәк. Нәтиҗәдә баба объектта һәм онык объектта ике бер исемдәге ысул эшләячәк, аларның алгоритмик нигезе төрле булачак, алар объектларга төрле үзлекләр бирәчәкләр. Бу объектлар полиморфизмы дип атала.

Object Pascal дә полиморфизм югарыда тасвирланган баба объект ысулларын мирас итеп алу һәм каплау механизмы ярдәмендә генә түгел, ә бәлки, күрсәтелгәнчә, баба ысулларына онык ысулларын чакыру мөмкинлеге бирүче виртуализация кулланып та үтәлә. 1 нче мисалны аңлатучы рәсемне карыйк:

TPet
name
@TPet.Speak
@TPet.PushPet

TDog
name
@TDog.Speak
@TDog.PushPet

TCat
name
@TCat.Speak
@TCat.PushPet

TBird
name
@TBird.Speak
@TBird.PushPet

Бер классның барлык объектлары өчен виртуаль ысуллар таблицасы бер.

2 нче мисал.

```
program Project1;
{$APPTYPE CONSOLE}
uses
SysUtils;
Type
    Transport=class
Name:String;
constructor Create(NameI:String);
procedure PlaySignal; virtual;
procedure Info; virtual;
end;
Type
    TVelo=class(Transport)
wheels:Integer;          // Тәгәрмәчләр саны
constructor Create(NameI:String;Wh:Integer);
procedure PlaySignal;override;
end;
Type
    TMoto=class(Transport)
Cylinders:Integer;
constructor Create(NameI:String;Cyl:Integer);
procedure PlaySignal;override;
end;
Type
    TAuto=class(Transport)
doors : Integer;
```

```
constructor Create(NameI:String;d:Integer);
procedure PlaySignal;override;
end;
constructor Transport.Create(NameI:String);
begin
Name := NameI;
end;
constructor TVelo.Create(NameI:String;Wh:Integer);
begin
Name := NameI;
wheels := Wh;
end;
constructor TMoto.Create(NameI:String;Cyl:Integer);
begin
Name := NameI;
Cylinders:=Cyl;
end;
constructor TAuto.Create(NameI:String;d:Integer);
begin
Name := NameI;
doors := d;
end;
procedure Transport.PlaySignal;
begin
writeln('!!!!!!!');
end;
procedure TVelo.PlaySignal;
begin
writeln('Din Din');
end;
procedure TMoto.PlaySignal;
begin
writeln('Bap Bap!');
end;
procedure TAuto.PlaySignal;
begin
writeln('Bee Bip ');
end;
```

```
procedure Transport.Info;
begin
writeln('Transport: ',Name);
PlaySignal;
end;
Var p : Array [1..4] of Transport;
i : Integer;
begin
p[1]:= TVelo.Create('Sputnik',2);
p[2]:= TMoto.Create('Ural',2);
p[3]:= TAuto.Create('Lada',4);
p[4]:= TAuto.Create('Niva',2);
for i := 1 to 4 do
p[i].Info;
readln;
end.
```

3.6.2. Берничә конструкторлы класслар

Берничә конструкторлы класс үрнәге итеп сызыкча булмаган тигезләмәләрне урталай бүлү ысулы, итерацияләр ысулы һәм Ньютон ысулы кулланып чишүне карыйк.

```
program Project_1;
{$APPTYPE CONSOLE}

                 { Тигезләмәләр чишү өчен класс }
Const
MaxN=1000;          // Ысул адымнарының максималь саны
Type TFunct=function(x:double):double;
Type
  TEquation=class
f: TFunct;          //  f(x)=0
g: TFunct;
                 {x-g(x)=0 һәм f(x)=0 эквивалент }
Df: Tfunct;         // f(x) ) функциясенең чыгарылмасы
a,b: double;        //  кисемтә чикләре
x0: double;         // тамырның якынча кыйммәте
eps: double;        // исәпләү төгәллеге
N: Integer;    { Якынайту ысулы адымнары саны
```

```
                    барлык ысуллар өчен бер уртак конструктор }
    constructor
Create(f_,g_,Df_:TFunct;a_,b_,x0_,eps_:double);overload;
                    { урталай бүлү ысулы өчен конструктор }
    constructor Create(f_:TFunct;
    a_,b_,eps_:double);overload;

                    { итерацияләр ысулы өчен конструктор }
    constructor Create(f_,g_:TFunct;
    x0_,eps_:double);overload;

                    { Ньютон ысулы өчен конструктор }
    constructor Create(x0_,eps_:double;
    f_,Df_:TFunct);overload;
    destructor Destroy;
    function Bisect:double;
    function Iterat:double;
    function Newton:double;
    function GetN:Integer;
    end;
    constructor TEquation.Create(f_,g_,Df_:TFunct;
    a_,b_,x0_,eps_:double);
                    { уртак конструктор }
    begin
    f:=f_; g:=g_; Df:=Df_;
    a:=a_; b:=b_; x0:=x0_; eps:=eps_;
    end;
    constructor TEquation.Create(f_:TFunct;
    a_,b_,eps_:double);

                    { урталай бүлү ысулы өчен конструктор }
    begin
    f:=f_;
    a:=a_; b:=b_; eps:=eps_;
    end;
    constructor TEquation.Create(f_,g_:TFunct;
    x0_,eps_:double);
```

{ итерацияләр ысулы өчен конструктор}

```
begin
f:=f_; g:=g_;
x0:=x0_; eps:=eps_;
end;
constructor TEquation.Create(x0_,eps_:double;
f_,Df_:TFunct);
```

{ Ньютон ысулы өчен конструктор }

```
begin
begin
f:=f_; Df:=Df_;
x0:=x0_; eps:=eps_;
end;
destructor TEquation.Destroy;
begin
end;
```

{ урталай бүлү ысулы }

```
function TEquation.Bisect:double;
Var u, v, f1, f2 : double;
begin
N := 1;
u := a; v := b;
f1 := f(u);
repeat
x0 := (u + v) * 0.5;
f2 := f(x0);
if  f2 = 0 then break;
if  f1*f2 >0
then
begin
u := x0; f1 := f2;
end
else v := x0;
Inc(N);
until (v - u) < eps;
Result := x0;
end;
```

```
                { итерацияләр ысулы }
function TEquation.Iterat:double;
Var x,t:double;
begin
N := 1;
repeat
x := g(x0);
t := abs(x-x0);
Inc(N);
x0 := x;
until (t<eps) Or (N>MaxN);
Result := x;
end;
                { Ньютон ысулы }
function TEquation.Newton:double;
Var t,x:double;
begin
N := 1;
repeat
x := x0-f(x0)/df(x0);
t := abs(x-x0);
Inc(N);
x0 := x;
until (t<eps) Or (N>MaxN);
Result := x;
end;

function TEquation.GetN:Integer;
begin
Result := N;
end;
                { Тигезләмә мисалы }
function f(x:double):double;     // тигезләмәнең сул ягы
begin
Result := x*x*x - 2*x - 5;
end;
function g(x:double):double;
                { x=g(x) һәм f(x)=0 бердәй, бертөрле көчле,
```

```
                итерацияләр ысулында кулланыла }
begin
Result := -x*x*x*0.04 + 1.08*x + 0.2;
end;
function df(x:double):double;
                { f(x) функциясе чыгарылмасы
                Ньютон ысулында кулланыла }
begin
Result := 3*x*x - 2;
end;
Var Eq: TEquation;
Begin
Eq := TEquation.Create(f,2,3,0.000001);
Writeln(' Bisect ',Eq.Bisect,' ',Eq.GetN,' ');
Eq := TEquation.Create(f,g,2.5,0.000001);
Writeln(' ',Eq.Iterat, ' ',Eq.GetN,' ');
Eq:= TEquation.Create(2.5,0.000001,f,df);
Writeln(' ',Eq.Newton,' ',Eq.GetN,' ');
Readln;      // тоткарлау
End.
```

Бу мисалда берничә конструктор кулланыла, өстәвенә шул мөһим, аларның исемнәре бертөрле. Delphi да бертөрле исемдә бердән артык функция яки процедура игълан итәргә ярый. Мондый функцияләр яңадан йөкләнгән (overloading) дип атала. Бу очракта функцияләрне overload директивасы белән тәэмин итәргә кирәк, ул функция башламының иң ахырында урнаштырыла, нокталы өтер белән аерылып алына. Функцияләрнең сигнатуралары төрле булырга тиеш. Бу функцияләрнең параметрлары исемлегендә параметрлар саны яки параметр типлары исемлеге төрле булырга тиеш дигән сүз. Гадәти функцияләр һәм процедуралар да, класс ысуллары да яңадан йөкләнгән була ала.

Delphi белешмә хезмәтеннән мисаллар

```
function Divide(X, Y: Real): Real; overload;
begin
Result := X/Y;
end;
function Divide(X, Y: Integer): Integer; overload;
```

```
begin
Result := X div Y;
end;
function Func(X: Real; Y: Integer): Real; overload;
...
function Func(X: Integer; Y: Real): Real; overload;
```

Түбәндәгн код хата китереп чыгарачак

```
function Cap(S: string): string; overload;
...
procedure Cap(var Str: string); overload;
```

Килешү буенча параметрлар һәм яңадан йөкләү

Килешү буенча параметрларны кулланганда яңадан йөкләү очрагында проблемалар килеп чыгуы мөмкин.

Мисаллар. (Delphi белешмә хезмәтеннән)

```
procedure Confused(I: Integer); overload;
procedure Confused(I: Integer; J: Integer = 0); overload;
Confused(X);            // Кайсы процедура чакырыла?
```

Бу код компиляция этабында хатага китерәчәк

3.7. Яңадан йөкләнә ала торган методлар

Ысуллар, гадәти функцияләр кебек үк, яңадан йөкләнә ала торган була алалар. Мондый ысулларның башламнары ахырына **overload** директивасы өстәргә кирәк. Яңадан йөкләнә ала торган ысулларның сигнатуралары төрле булырга тиеш.

Әгәр виртуаль ысуллар яңадан йөкләнә икән, чыгарылма классларда бу ысуллар өстәмә reintroduce директивасы белән тәэмин ителәләр.

Мисал. (Delphi белешмә хезмәтеннән).

```
type
T1 = class(TObject)
procedure Test(I: Integer); overload; virtual;
end;
T2 = class(T1)
procedure Test(S: string); reintroduce; overload;
end;
...
SomeObject := T2.Create;
```

SomeObject.Test('Hello!'); // calls T2.Test
SomeObject.Test(7); // calls T1.Test
published өлкәсеннән ысулларны яңадан йөкләргә ярамый.

Delphi белешмә хезмәтеннән мисал.

```
type
TSomeClass = class
published
function Func(P: Integer): Integer;
function Func(P: Boolean): Integer // error
```

3.8. Ысулларны делегирлау

Ысулларга күрсәткечләр процедур типлар кебек үк тасвирлана. Бердәнбер аерма булып формаль параметрлар исемлегеннән соң of object ачкыч сүзләрен күрсәтү тора.

```
type
  TMyMethod = procedure (Sender : Object) of object;
```

Процедур типлар һәм ысул типларының уртак һәм аермалы якларын күрсәтү өчен мисал карыйк, анда процедур типларны караган мисалдагы функцияләр кулланыла.

```
unit Unit2;
        {Кулланучы стандарт библиотекасының прообразы}

interface

type
  TMyFunc = function ( X : Integer ) : Real of object;

  TMyClass1 = class
    private
      FField : Real;
      FMyFunc: TMyFunc;
    protected
      function FirstFunc ( X : Integer ) : Real;
      procedure SetField ( Value : Real );
    public
      procedure TakeAndSet ( X : Integer );
      property Field : Real
```

```
      read FField  write SetField;
    property MyFunc : TMyFunc
      read FMyFunc write FMyFunc;
  end;

var
  MyObject1 : TMyClass1;

implementation

function TMyClass1.FirstFunc ( X : Integer ) : Real;
begin
  Result := SQR ( X )/2;
end;

procedure TMyClass1.SetField ( Value : Real );
begin
  FField := Value
end;

procedure TMyClass1.TakeAndSet ( X : Integer );
begin
  SetField (MyFunc(X));
end;

initialization
  MyObject1 := TMyClass1.Create;
  MyObject1.MyFunc := MyObject1.FirstFunc;

finalization
  MyObject1.Free;

end.
unit Unit1;
```

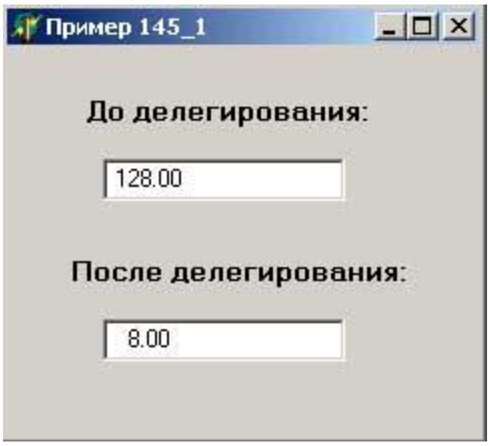
{Unit2 стандарт модулен кулланучы һәм «үзенең» SecondFunc ысулын Unit2 дән импортланучы MyObject1 объектына күчерүче кушымта.}

```
interface
```

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Unit2;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TMyClass2 = class
      function SecondFunc ( X : Integer ) : Real;
  end;


var
  Form1: TForm1;
  MyObject2 : TMyClass2;

implementation

{$R *.dfm}

function TMyClass2.SecondFunc ( X : Integer ) : Real;
begin
  Result := SQRT ( X )*2;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
 St: String;
begin
 with MyObject1 do
 begin
  TakeAndSet(16);    // 128 кыйммәтен билгели
  Str (Field:7:2, St);
  Edit1.Text := St;
  MyObject2:= TMyClass2.Create;
  MyFunc:= MyObject2.SecondFunc;   //  Күчерү
  TakeAndSet(16);          //  8 кыйммәтен билгели
  Str (Field:7:2, St);
  Edit2.Text:= St;
  MyObject2.Free;
 end;
end;

end.
```

Нәтиҗә:



Рәс. 28. Кушымта эше нәтиҗәсе

Бу мисалда ике модуль тасвирланган: Unit2 дә кулланучы библиотекасының беренчел рәвеше, Unit1 дә бу библиотеканы кулланучы кушымтаның беренчел рәвеше бар. Unit2 модулендә TMyClass1 классы игълан ителгән, аның эчендә процедур типлы MyFunc үзлеге бар, бу классның MyObject1 объекты игълан ителгән,

ул модульнең инициализация өлешендә үзенең FirstFunc эчке ысулын куллануга стандарт рәвештә көйләнә. Нәтиҗәдә TMyClass1 классының MyFunc үзлегенә бирелгән ысулны чакыручы TakeAndSet ысулы килешү буенча FirstFunc ысулын чакырачак. Тик MyFunc үзлегенең процедур тибы ярдәмендә библиотеканы кулланучы кушымтада TakeAndSet ысулын TMyClass2 классыннан SecondFunc ысулын чакыра торган итеп яңадан көйләп була.Ул тип буенча FirstFunc ысулы белән бердәй. Бер класста тасвирланган гамәлне икенче класс ысулына тапшыруны күчерү (делегирование) дип атыйлар. Бу мисалда күчерүне оператор башкара:

MyFunc := MyVar2.SecondFunc;

Delphi вакыйгалары программа башкарылган вакытта үзгәртеп була торган процедур үзлекләр рәвешендә реализациялэнгэнгә алар өчен шулай ук күчерү кулланып була. Delphi ның стандарт вакыйгасына үз ысулын күчерү өчен бу вакыйганы эшкәртүче ысулның башлам рәвешен белергә кирәк. Мәсәлән, TNotifyEvent тибы өстәмә параметрлары булмаган вакыйгалар өчен тасвирланган.

type
TNotifyEvent = procedure (Sender: TObject) of object;

Типта тасвирланган бердәнбер Sender параметры барлык вакыйгаларны эшкәртүчеләрнең стандарт параметры булып тора.

3.9. Үзлекләр

Бу класс элементы кырларга мөрәҗәгать итүне, керүне көйли.

Синтаксис.

Property Имя:Тип
read Имя_метода_чтения
write Имя_метода_записи

Үзлекне игълан итү махсус property сүзе ярдәмендә башкарыла. read һәм write ачкыч сүзләре мөрәҗәгать итү, керү спецификаторлары дип атала. read сүзеннән соң үзлекнең кыйммәтен укыганда (алганда) мөрәҗәгать ителә торган кыр яки ысул күрсәтелә. write сүзеннән соң үзлекнең кыйммәтен язганда (билгеләгәндә) мөрәҗәгать ителә торган кыр яки ысул күрсәтелә.

Үзлек исемнәре кыр исемнәре белән тәңгәл килмәсен өчен кырларны F хәрефеннән башлыйлар (инглизчә field).

Үзлекләрнең кыйммәтләрен алу (уку) һәм билгеләү (язу) ысуллары аерым кагыйдәләргә буйсына. Үзлекне уку ысулы – үзлек тибындагы кыйммәтне кайтаручы функция. Үзлекне язу ысулы – үзлек тибындагы параметрны алучы процедура.

Мисал

```
program MyProject1;
{$APPTYPE CONSOLE}
uses
SysUtils;
Type
TaClass=class
FNumber:Integer;
function GetField:Integer;
procedure SetField(Value:Integer);
property Number:Integer
read GetField
write SetField;
end;
function TaClass.GetField:Integer;
begin
GetField:=FNumber;
end;
procedure TaClass.SetField(Value:Integer);
begin
FNumber:=Value;
end;
Var aClass:TaClass; Value:Integer;
begin
aClass:=TaClass.Create;
aClass.Number :=0;                  // Үзлеккә мөрәҗәгать итү
Value:= aClass.Number;        // Үзлеккә мөрәҗәгать итү
Writeln('Value=', Value);
Readln;
end.
```

Искәрмә. Мондый үзләштерү операторы да булырга мөмкин

aClass.FNumber := 0;

Аерма шунда, түбәндәге оператор

aClass.Number :=0;

aClass.SetField(0); ысулын чакыруга китерә, ә беренче оператор турыдан-туры кыйммәтне билгели. Гомумән, үзлекләрне кыйммәтләрне үзгәрткәндә өстәмә гамәлләр, мәсәлән, типларны үзгәртү кирәк булганда кулланалар. Бу гамәлләр үзлекнең read һәм write өлкәләрендә күрсәтелгән ысулларга языла.

Мисал (Delphi Help тан).

```
type
THeading = 0..359;
TCompass = class(TControl)
private
FHeading: THeading;
procedure SetHeading(Value: THeading);
published
property Heading: THeading read FHeading write SetHeading;
...
end;
```

Бу вакытта түбәндәге код

```
if Compass.Heading = 180 then GoingSouth;
Compass.Heading := 135;
```

Автомат рәвештә бу кодка алмашына

```
if Compass.FHeading = 180 then GoingSouth;
Compass.SetHeading(135);
```

SetHeading ысулында гади билгеләүгә караганла күпкә катлаулырак код була ала мәсәлән, менә мондый

```
procedure TCompass.SetHeading(Value: THeading);
begin
if FHeading <> Value then
begin
FHeading := Value;
Repaint; // update user interface to reflect new value – яңа кыйммәтне күрсәтү өчен кулланучы интерфейсын яңарту
end;
end;
```

Әгәр мөрәҗәгать итү, керү спецификаторларының берсе төшереп калдырылган булса, үзлекнең кыйммәтен укып кына (read спецификаторы бирелсә) яки язып кына (write спецификаторы бирелсә) була.

Кырлардан аермалы буларак үзлекләрнең хәтердә адреслары юк, шуңа аларга @ операциясен куллану тыела. Нәтиҗә буларак, аларны процедуралар һәм функцияләрнең var- һәм out-параметрларында тапшырырга ярамый.

Delphi мохитендә объектка-ориентлашкан программалау технологиясе кырларга турыдан-туры мөрәҗәгать итүдән тыелырга куша, аның урынына тиешле үзлекләр төзү кирәк.

3.10. Үзлекләр массивлары

Үзлекләр массивлары – ул индекслы үзлекләр.

Үзлекләр массивларын игълан итү.

Индексларның исемнәреннән һәм типларыннан торучы параметрлар исемлеген кертергә кирәк. Мәсәлән, болай:

property Objects[Index: Integer]: TObject read GetObject write SetObject;

property Pixels[X, Y: Integer]: TColor read GetPixel write SetPixel;

property Values[const Name: string]: string read GetValue write SetValue;

Мисал

```
program MyProject2;
{$APPTYPE CONSOLE}
uses
SysUtils;
type
TMyArray = class
FItems: array of string;
ItemCount:Integer;
constructor Create(NMax:Integer);
function GetItem(Index: Integer): string;
property Items[Index: Integer]: string read GetItem;
end;
function TMyArray.GetItem(Index: Integer): string;
begin
Result := FItems[Index];
end;
constructor TMyArray.Create(NMax:Integer);
Var i:Integer;
begin
```

```
SetLength(FItems,NMax);              // Хәтер бүленеп бирелә
ItemCount:=NMax;
for i:= 0 to NMax-1 do
FItems[i] := 'Line ' + IntToStr(i);
end;
Var R: TMyArray;
I: Integer;
begin
R:=TMyArray.Create(10);
for I := 0 to R.ItemCount - 1 do
Writeln(R.Items[I]);     // Үзлеккә мөрәҗәгать итү
Readln;
end.
```

Уку (read) һәм язу (write) ысулларында параметрлар сыйфатында индекслар исемлеге бар. Мәсәлән,

```
function GetObject(Index: Integer): TObject;
function GetPixel(X, Y: Integer): TColor;
function GetValue(const Name: string): string;
procedure SetObject(Index: Integer; Value: TObject);
procedure SetPixel(X, Y: Integer; Value: TColor);
procedure SetValue(const Name, Value: string);
```

Бу үзлекләргә мөрәҗәгать итү болай башкарыла:

```
if Collection.Objects[0] = nil then Exit;
Canvas.Pixels[10, 20] := clRed;
Params.Values['PATH'] := 'C:\BIN';
```

Бу очракта чынлыкта тиешле ысуллар чакырыла

```
if Collection.GetObject(0) = nil then Exit;
Canvas.SetPixel(10, 20, clRed);
Params.SetValue('PATH', 'C:\BIN');
```

Үзлек-массив күп үлчәмле булырга мөмкин. Бу очракта уку һәм язу ысулларының тиешле типтагы индекслы параметрлары үзлек-массивныкы кадәр булырга тиеш.

Үзлек-массивны бирелгән класс объектларының төп үзлеге итеп була. Моның өчен үзлек тасвирламасына default сүзе өстәлә. Мәсәлән:

```
property Items[Index: Integer]: string read GetItem; default;
```

Items үзлеген болай игълан итү безгә класс объектының үзен массив буларак карарга һәм программадан мөрәҗәгать иткәндә үзлек-

массивның исемен төшереп калдырырга мөмкинлек бирә. Мәсәлән, Writeln(R.Items[I]) ; урынына Writeln(R[I]);.

Мисал.

```
program MyProject3;
{$APPTYPE CONSOLE}
uses
SysUtils;
type
TMyArray = class
FItems: array of string;
ItemCount:Integer;
constructor Create(NMax:Integer);
function GetItem(Index: Integer): string;
property Items[Index: Integer]: string read GetItem;
default;
end;
function TMyArray.GetItem(Index: Integer): string;
begin
Result := FItems[Index];
end;
constructor TMyArray.Create(NMax:Integer);
Var i:Integer;
begin
SetLength(FItems,NMax);   // Үлчәмне билгеләү
ItemCount:=NMax;
for i:= 0 to NMax-1 do
FItems[i] := 'Line ' + IntToStr(i);
end;
Var R: TMyArray;
I: Integer;
begin
R:=TMyArray.Create(10);
R.Items['Line 2'] := 'Example';
for I := 0 to R.ItemCount - 1 do
Writeln(R[I]);       // Үзлеккә мөрәҗәгать итү
Readln;
end.
```

Бер үк ысул бер типның берничә үзлегенең кыйммәтләрен алу (билгеләү) өчен кулланылырга мөмкин. Бу очракта һәр үзлеккә бөтен санлы индекс билгеләнә, ул уку (язу) ысулына беренче параметр булып бирелә. index ачкыч сүзе төрле үзлекләр өчен бер үк ысулны чакыруны оештыру өчен кулланыла.

Rectangle объекты TRectangle классына керсә, түбәндәге оператор

Rectangle.Right := Rectangle.Left + 100;

бу операторга тиңдәш

Rectangle.SetCoordinate(2,Rectangle.GetCoordinate(0)+ 100);

Беренче карашка үзлек-массивларның гади үзлекләрдән аермасы «массив» тибындагы үзгәрешлеләрнең гади типтагы үзгәрешледән аермасына ошаган (ягъни индексның булуы белән). Тик үзлек-массивларда индексларны тасвирлау һәм куллану ысулы гади массивларны индекслаудан күпкә аерыла. Үзлек-массивларның гади үзлекләрдән һәм гади массивлардан аермаларына аерым тукталыйк.

Үзлек-массивларның гади үзлекләрдән аермасы:

1. Игълан итүнең төп үзенчәлеге булып индекслы параметрлар исемлеге булу тора. Игътибар итик, индекслар түгел, ә индексларны билгеләүче параметрлар, алар процедураларның формаль параметрлары кебек типны күрсәтеп языла. Индекслы параметрлар исемлеген тасвирлаганда түгәрәк түгел, ә бәлки квадрат җәяләр кулланыла. Монда үзлек-массивларның гади үзлекләрдән дә, гади массивлардан да аермасы күренә.

2. read һәм write мөрәҗәгать итү, керү спецификаторларын тасвирлауда ысулларның идентификаторлары гына күрсәтелә ала. Кыр идентификаторын турыдан-туры куллану тыела.

3. Уку спецификаторында күрсәтелгән ысул функция булырга тиеш, аның параметрлары саны, тибы һәм урнашу тәртибе үзлекнең индекслы параметрлары тасвирламасына тиңдәш булырга тиеш. Бу функция кайтара торган нәтиҗә тибы үзлек тибы белән тәңгәл килергә тиеш.

4. Язу спецификаторында күрсәтелгән ысул процедура булырга тиеш, аның башлангыч параметрлары саны, тибы һәм урнашу тәртибе үзлекнең индекслы параметрлары тасвирламасына тиңдәш булырга тиеш. Моннан тыш, процедураның параметрлар исемлегендә үзлек тибындагы ук бер өстәмә параметр булырга тиеш, ул яңа кыйммәт

тапшыру өчен кирәк. Бу параметр параметр-кыйммәт яки параметр-константа була ала.

5. Үзлек-массивлар published була алмый.

Үзлек-массивларның «массив» тибындагы гади үзгәрешлеләрдән аермасы:

1. Үзлек-массивка бер бөтен буларак мөрәҗәгать итеп булмый. Аның аерым элементлары белән генә эш итәргә рөхсәт ителә.

2. Үзлек-массивларның индекслары тибы тәртип кенә түгел, теләсә нинди була ала.

Тагын бер мисалда үзлек-массивларны куллануны карыйк.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Memo2: TMemo;
    Label1: TLabel;
    Label2: TLabel;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TVectorClass = class
  private
    FVector : array [1..10] of Word;
  protected
    function GetVector (Index : Integer) : Word;
    procedure SetVector (Index : Integer; Value : Word);
```

```
    procedure AddToVectorElem (Index : Integer; Value : Word);
  public
    property Elements [Index : Integer] : Word
        read GetVector
        write SetVector;    default; { килешү буенча үзлек}
    property AddToElement [Index : Integer] : Word
        write AddToVectorElem;
  end;


var
  Form1: TForm1;
  MyVector : TVectorClass;

implementation

{$R *.dfm}

function TVectorClass.GetVector(Index : Integer) : Word;
begin
  Result := FVector [Index];
end;

procedure TVectorClass.SetVector (Index : Integer; Value : Word);
begin
  FVector [Index] := Value;
end;

procedure TVectorClass.AddToVectorElem(Index : Integer; Value : Word);
begin
  FVector [Index] := FVector [Index] + Value;
end;


procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
```

```
  S : String;
begin
  MyVector := TVectorClass.Create;
  for i := 1 to 10 do
    MyVector[i] := i;
        { Elements үзлек исеме төшереп калдырылырга мөмкин,
        чөнки бу үзлек килешү буенча кабул ителәчәк}
  for i := 1 to 10 do
  begin
    Str(MyVector.Elements[i], S);
    Form1.Memo1.Lines.Add(S);
  end;
  for i := 1 to 10 do
    MyVector.AddToElement[i] := 5;
        { AddToElement үзлек исеме һәрвакыт тулысынча
        күрсәтелергә тиеш }
  for i := 1 to 10 do
  begin
    Str (MyVector[i], S);
    Form1.Memo2.Lines.Add(S);
  end;
  MyVector.Free;
end;

end.
```

Барлык санап үтелгән аермаларга карамастан, үзлек-массив элементларына мөрәҗәгать итү гади массив элементларына мөрәҗәгать иткән кебек башкарыла.

Моннан тыш, югарыда әйтеп үтелгәнчә, default директивасы үзлек-массивларда саклау спецификаторы булып түгел, ә үзлек-массивларның кайсы килешү буенча алыначакны күрсәткеч булып тора. Каралган мисалда – бу Elements үзлеге. Мисалның кодыннан күренгәнчә, Elements үзлегенә ике юл белән мөрәҗәгать итеп була:

- теләсә нинди башка үзлеккә кебек үк тулы MyVector.Elements[i] исеме белән;
- килешү буенча алынган үзлеккә кебек үк, объектның исемен һәм индексын гына күрсәтеп: MyVector[i].

Массивлар кебек үк, үзлек-массивлар күп индекслы була ала. Мондый үзлекләрнең индекслы параметрлары исемлекләрендә һәм аларга мөрәҗәгать итү, керү ысулларының формаль параметрлары исемлекләрендә үлчәмгә ярашлы индекслар саны булырга тиеш. Киләсе мисалда ике үлчәмле үзлекне куллану күрсәтелә. А һәм К хәрефләре белән тактада шашкаларның беренчел урнашуы модельләштерелә.

```
unit Unit1;
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  TRow  ='a'..'h' ;
  TColumn  = 1..8;
  TCheckersBoard =   class
  private
    FBoard : array [TRow,TColumn] of  Char;
  protected
    function GetChecker (Row: TRow; Col: TColumn):Char;
    procedure  SetChecker (Row: TRow; Col: TColumn; const Value:
Char);
  public
    property Board [Row: TRow; Col: TColumn]: Char
      read GetChecker write SetChecker; default;
  end;
```

```
var
  Form1: TForm1;
  MyBoard : TCheckersBoard;


implementation

{$R *.dfm}

function TCheckersBoard.GetChecker (Row : TRow; Col :TColumn)
: Char;
begin
  Result := FBoard [Row,Col];
end;

procedure TCheckersBoard.SetChecker (Row : TRow; Col :
TColumn;
          const Value : Char) ;
begin
  FBoard [Row, Col] :=Value;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i : TRow;
  j : TColumn;
  s : String;
  k,l : Integer;
begin
  MyBoard := TCheckersBoard.Create;
  for i := 'a' to 'h' do
    for j :=1 to 8 do
      MyBoard [i,j]:=' ';
  Memo1.Lines.Clear;
  k:=0;
  for i := 'a' to 'h' do
  begin
```

```
    if Ord(i) mod 2 = 1 then
    begin
      MyBoard [i,1]:='A';
      MyBoard [i,3]:='A';
      MyBoard [i,7]:='K';
    end
    else
    begin
      MyBoard [i,2]:='A';
      MyBoard [i,6]:='K';
      MyBoard [i,8]:='K';
    end;
  end;
  for j :=1 to 8 do
  begin
    S:='';
    for i := 'a' to 'h' do
         S:=S+MyBoard [i,j];
    Memo1.Lines.Add(S);
  end;
end;

end.
```

3.11. Индекс спецификаторлары

Әгәр дә үзлекнең синтаксисын алсак, тагын бер спецификаторны куллану каралмаган икәне күренә. Бу index спецификаторы, ул индекс спецификаторы дип атала. Индекс спецификаторлы үзлекләр һәм үзлек-массивларның төрле төшенчәләр булуын әйтеп үтик. Мондый үзлекләр үзлекләрнең аерым очрагы булып тора, һәм үзлек-массивлар кулланылган максатта ук кулланыла ала, тик кайчак күркәмрәк күренәләр.

Индекс спецификаторлы үзлекне куллану мисалын китерик.

```
unit Unit1;

interface

uses
```

```
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

  Masks  =  class
  private
    FMasks:  array[1..3] of Word;
    function GetMask(Index: Integer): Word;
    procedure SetMask(Index: Integer; Value: Word);
  public
    property MaskForTask1: Word  index 1 read GetMask write SetMask;
    property MaskForTask2: Word  index 2 read GetMask write SetMask;
    property MaskForTask3: Word  index 3 read GetMask write SetMask;
  end;

var
  Form1: TForm1;
  MyMasks: TMasks;

implementation

{$R *.dfm}

function TMasks.GetMask;
begin
```

```
  Result := FMasks [Index]
end;

procedure TMasks.SetMask;
begin
  FMasks[Index] := Value
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  St : String;
  i : Integer;
begin
  MyMasks := TMasks.Create;
  Memo1.Lines.Clear;
  with MyMasks do
  begin
    MaskForTask1 := 10;
    MaskForTask2 := 5;
    MaskForTask3 := MaskForTask1 and MaskForTask2;
                Соңгы үзләштерү операторы SetMask (3,
          GetMask(1) and GetMask (2)); операторына эквивалент

                { Memo1 кырына маска кыйммәтләрен чыгару}

 St := 'MaskForTask1 = '+IntToStr(MaskForTask1);
    Memo1.Lines.Add(St);
    St := 'MaskForTask2 = '+IntToStr(MaskForTask2);
    Memo1.Lines.Add(St);
    St := 'MaskForTask3 = '+IntToStr(MaskForTask3);
    Memo1.Lines.Add(St);
  end;
end;
end.
```

Мисалдан күренгәнчә, уникаль исемле, төрле берничә үзлек бер үк кыр-массивның төрле элементларына мөрәҗәгать итү, керү өчен бер үк уку һәм язу ысулларны кулланалар. Башкача әйткәндә,

мөрәҗәгать итү, керү ысулларының автомат көйләнүе бара, ул бу вакытта ысулларның кайсы үзлеккә хезмәт күрсәтүләренә бәйле.

Индекс спецификаторны куллану үзлекне тасвирлау формасына куя торган берникадәр чикләүләрне билгеләп үтик:

- read һәм write мөрәҗәгать итү, керү спецификаторлары тасвирламаларында, үзлек-массивлардагы кебек, ысул идентификаторлары гына күрсәтелә ала. Кыр идентификаторын турыдан-туры куллану тыела;
- үзлекнең уку һәм язу ысулларында индексны тапшыру өчен өстәмә Integer параметр-кыйммәте булырга тиеш;
- уку ысулында өстәмә индекс параметры тапшырылучы параметрлар исемлегендә соңгы булырга тиеш;
- язу ысулында өстәмә индекс параметры нәкъ үзлекнең яңа кыйммәтен билгеләүче параметр алдыннан барырга тиеш тапшырылучы параметрлар исемлегендә соңгы булырга тиеш;
- үзлеккә мөрәҗәгать иткәндә index спецификаторы белән билгеләнүче индекс автомат рәвештә өстәмә параметрга факттагы параметр сыйфатында тапшырыла;
- index директивасы артында торучы константа бөтен сан булырга һәм аның кыйммәте -32767 дән 32767 гә кадәр булырга тиеш.

3.12. Класс ысуллары һәм класс күрсәткечләре

Класс ысуллары классның объектлары белән түгел, ә классның үзе белән эш итә. Бу ысулларның башламнарын тасвирлау өчен махсус class сүзе кулланыла, ул procedure һәм function сүзләре алдыннан куела.

Класс ысулы класс исеменә сылтама ярдәмендә яки аерым объект исеменә сылтама ярдәмендә чакырылырга мөмкин. Икенче очракта объект классы Self параметры кебек тапшырыла.

Мисал.

```
program Project1;

type
  TMyFirstClass = class
    class function GetParentName: string;
  end;

  TMySecondClass = class (TMyFirstClass)
```

```
  end;

var
  MyVar: TMySecondClass;
  S: string;

class function TMyFirstClass.GetParentName: string;
begin
  GetParentName := Self.ClassParent.ClassName
end;
begin
  S := TMyFirstClass.GetParentName;

                    {класска сылтама ясап чакыру}
  Writeln ('The Parent of TMyFirstClass is ', S);
  S := TMySecondClass.GetParentName;

                    { класска сылтама ясап чакыру }
  Writeln ('The Parent of TMySecondClass is ', S);
  MyVar := TMySecondClass.Create;
  S := MyVar.GetParentName; // объектка сылтама ясап чакыру
  Writeln ('The Parent of MyVar is ', S);
  MyVar.Free;
  Readln;
end.
```

«Классның үзе белән эш итә» дигән сүз нәрсә аңлатканын карап китик.

Программа башкарылган вакытта һәр класс өчен хәтердә аның тибы турында мәгълүмат саклана. Ул башкарылу вакыты тибы турындагы мәгълүмат дип атала (Run-Time Type Information, RTTI). Класс ысуллары бу мәгълүмат белән эшләү мөмкинлеген оештыру өчен кулланыла.

Искәрмә. RTTI объектлар өчен түгел, ә бәлки класслар өчен төзелә. Шуңа күрә бу мәгълүматка алдан объектлар төземичә мөрәҗәгать итеп була. Класс ысулларын гамәлгә ашыру аерым объектларның кыйммәтләренә бәйле булырга тиеш түгел.

RTTI белән эшләү өчен Object Pascalдә тагын бер ысул бар, ул классларга сылтамалар куллана, чынлыкта алар RTTIга сылтамалар

булып тора. Классларга сылтамаларны тасвирлау өчен махсус class of сүзләре кулланыла.

Мисал өчен консоль режимында эшләүче программа китерик. Анда классларга өч күрсәткеч бар: PtrMyFirstClass TPtrMyFirstClass классына күрсәтә, PtrMySecondClass TptrMySecondClass классына күрсәтә, PtrTObject TObject классына күрсәтә. TClass ның TObject классына күрсәткечләрнең алдан билгеләнгән тибы булуын искәртеп китик. Бу күрсәткечләр ярдәмендә ClassName класс ысулын чакыру башкарыла, ул TObject классында тасвирланган һәм үз классының исемен кайтара. Моннан тыш бу мисалда классларга күрсәткечләрнең ярашуы күрсәтелә, аның кагыйдәләре буенча баба класска күрсәткечкә бала класс адресларын биреп була, тик киресенчә булмый. Шуңа күрә төп TObject классына күрсәткеч ярдәмендә RTTI дан программада кулланылучы теләсә нинди классның исемен алып була.

```
program Project2;

type
  TMyFirstClass = class
  end;
  TMySecondClass = class (TMyFirstClass)
  end;
  TPtrMyFirstClass = class of TMyFirstClass;
  TPtrMySecondClass = class of TMySecondClass;

var
  PtrMyFirstClass : TPtrMyFirstClass;
  PtrMySecondClass : TPtrMySecondClass;
  PtrTObject : TClass;
  S : string;

begin
      {PtrTObject күрсәткече үзенең TObject классына да, аның токым класслары TMyFirstClass һәм TMySecondClass ка да күрсәтә ала.}

 PtrTObject := TObject;
  S := PtrTObject.ClassName;
```

```
  Writeln ('PtrTObject can point to class ', S);
  PtrTObject := TMyFirstClass;
  S := PtrTObject.ClassName;
  Writeln ('PtrTObject can point to class ', S);
  PtrTObject := TMySecondClass;
  S := PtrTObject.ClassName;
  Writeln ('PtrTObject can point to class ', S);
  Writeln;
      {PtrMyFirstClass күрсәткече үзенең TMyFirstClass
      классына да, аның токым классы TMySecondClass ка да
      күрсәтә ала.}
  PtrMyFirstClass := TMyFirstClass;
  S := PtrMyFirstClass.ClassName;
  Writeln ('PtrMyFirstClass can point to class ', S);
  PtrMyFirstClass := TMySecondClass;
  S := PtrMyFirstClass.ClassName;
  Writeln ('PtrMyFirstClass can point to class ', S);
  Writeln;
      {PtrMySecondClass күрсәткече үз классына гына күрсәтә
      ала. }
  PtrMySecondClass := TMySecondClass;
  S := PtrMySecondClass.ClassName;
  Writeln ('PtrMySecondClass can point to class ', S);
  Readln
end.
```

3.13. Класслар белән гамәлләр

Башкарылу вакыты типлары турындагы мәгълүмат (RTTI) белән эшләү өчен Object Pascal дә ике яңа операция is һәм as кертелгән.

is операциясе бинар, ягъни ике операндлы була, һәм түбәндәге рәвешне ала:

<Объект> is <Класс>.

Ул башкару вакытында Объектның Класска яки аның берәр токымына керү-кермәвен билгеләргә мөмкинлек бирә. Нәтиҗә булып буль зурлыгы тора, әгәр объект Класс классы белән үзләштерү буенча ярашлы булса, ул True кыйммәте ала, каршы очракта False кыйммәте ала. is операциясе чагыштыру операцияләре =, <>, <, >, <=, >= һәм in

операциясе белән беррәттән иң түбән приоритетлы, шуңа күрә аңлатмаларда үзенең операндлары белән бергә җәягә алына:

if (Sender is TEdit) or (Sender is TMemo) then

is операциясен куллануның иң еш очрагы булып типларны куркынычсыз үзгәртү тора, аның өчен түбәндәге рәвештәге конструкция кулланыла:

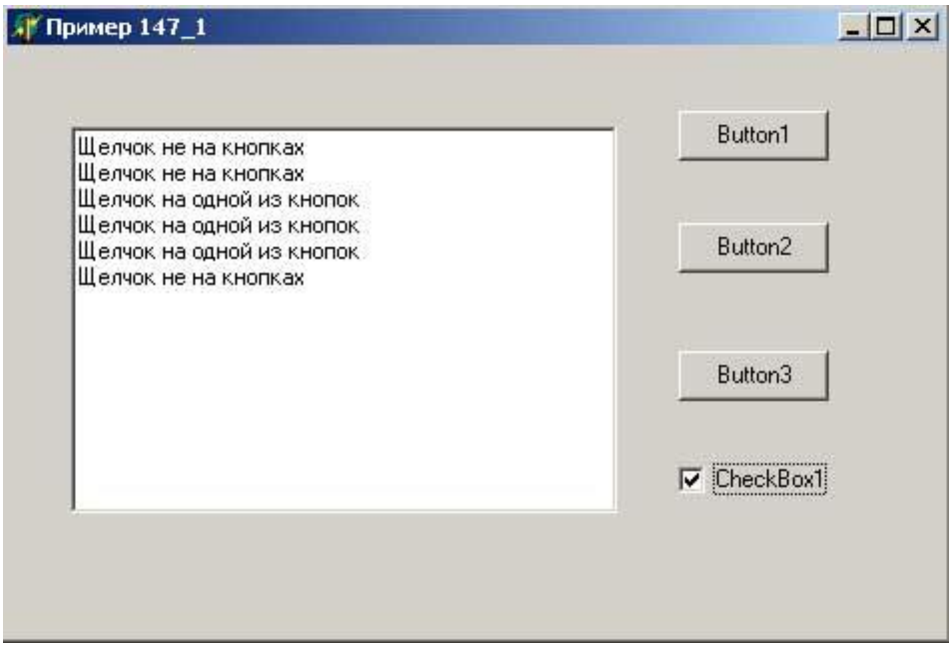
if <Объект> is <Класс> then <Класс> (<Объект>).<Метод>;

is операциясенең эшен күрсәтү өчен мисал китерик.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    CheckBox1: TCheckBox;
    procedure CommonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
```

```
procedure TForm1.CommonClick(Sender: TObject);
begin
  if Sender is TButton then
    Memo1.Lines.Add ('Щелчок на одной из кнопок')
  else
    Memo1.Lines.Add ('Щелчок не на кнопках')
end;

end.
```

Бу мисалда TButton классының төймәләре өчен уртак OnClick вакыйгасын эшкәртүчене кулланганда компонент-төймәләрне форманың барлык компонентлары күплегеннән аерып билгеләү күрсәтелгән. Мисал дөрес эшләсен өчен Object Inspector ярдәмендә Form1 формасының барлык компонентларына CommonClick идентификаторлы бер үк OnClick вакыйгасын эшкәртүчене тиңдәш итеп куярга кирәк. Проектны эшләтеп җибәргәннән соң бер-бер артлы Form1 формасына, Memo1 кырына, Button1, Button2, Button3 төймәләренә һәм контроль CheckBox1 кырына чиртелсә, түбәндәге нәтиҗә килеп чыга:



Рәс. 29. Кушымтаның эш нәтиҗәсе

as операциясе типларны үзгәртү өчен алдан билгеләнгән, аның is операциясе кебек үк ике операнды бар, һәм аның гомуми рәвешен түбәндәгечә күрсәтеп була:

<Объект> as <Класс> .

Бу операция тапкырлау тибындагы операцияләр (*, /, dlv, mod, and, shl һәм shr) белән беррәттән икенче приоритетка ия.

as операциянең нәтиҗәсе булып шул ук объект тора, тик ул үзенең башлангыч классына түгел, ә операторда күрсәтелгән Класс классына керә.

as операциясенең типларны турыдан-туры үзгәртүдән

<Класс>(<Объект>)

аермасы: башкарылу вакытында as операциясе эшкәртелгәндә типларны үзгәртүнең дөреслеге тикшерелә. Әгәр Объект Класс белән үзләштерү буенча ярашлы булса, үзгәртү башкарыла. Әгәр ярашлы булмаса, гадәттән тыш очрак килеп чыга, аның өчен кирәк булганда үз эшкәртүчесен язып була. Типларны турыдан туры үзгәрткәндә ярашлылыкка бернинди тикшерү үткәрелми.

as операциясе башкарылу вакытында типларны куркынычсыз үзгәртү өчен кулланыла һәм югарыда китерелгән is операторлы конструкциягә эквивалент. Тик аннан аермалы буларак, with операторы белән бергә кулланылса, җыйнаграк һәм күрсәтмәле код язарга мөмкинчелек бирә:

with <Объект> as <Класс> do

яки турыдан-туры идентификаторда:

(<Объект> as <Класс>).<Поле>

(<Объект> as <Класс>).<Метод>

as операциясе ярдәмендә типларны үзгәртүне күрсәтү өчен is операциясе өчен югарыда китерелгәнгә ошаган мисал алыйк..

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
```

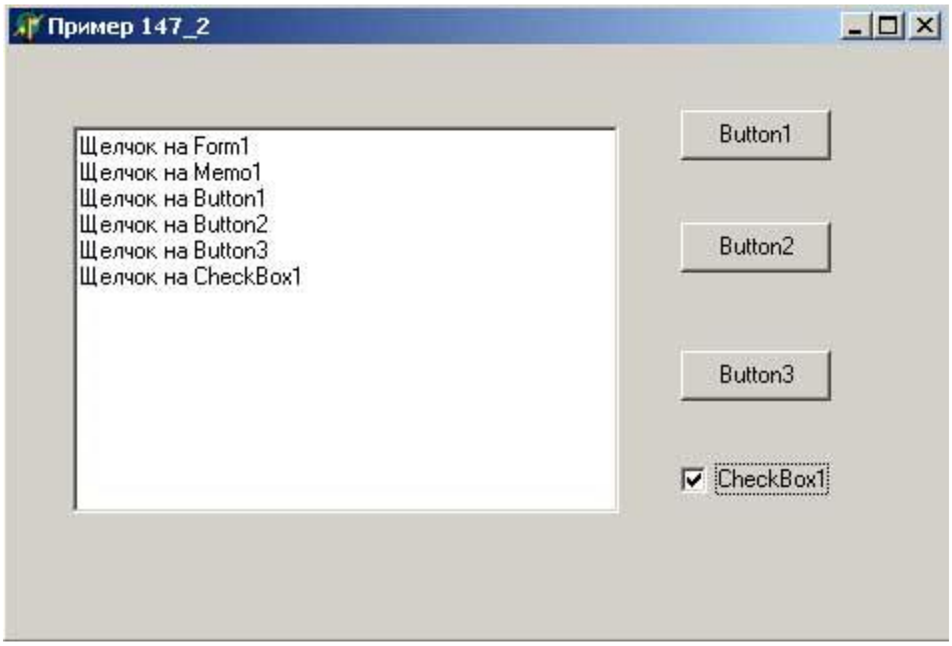
```
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    CheckBox1: TCheckBox;
    procedure CommonClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.CommonClick(Sender: TObject);
begin
  Memo1.Lines.Add ('Щелчок на ' +(Sender as  TControl).Name)
end;

end.
```

Form1 формасының барлык компонентлары өчен OnClick вакыйгасын эшкәртүче TForm1.CommonClick ысулында вакыйганың чыганак-объект тибын (Sender параметры) TControl класс тибына үзгәртү башкарыла. TControl классы проектта кулланылучы барлык компонентларның (Form1, Memo1, Button1, Button2, Button3, CheckBox1) нәсел башы булып торганга үзгәртү дөрес башкарыла. Нәтиҗәдә бер үк оператор:

Memo1.Lines.Add ('Щелчок на ' + (Sender as TControl) .Name);

Memo1 кырына тычкан белән чирткән барлык компонентлар исемнәрен чыгара. 30 нчы рәсемдә проект эшенең әгәр аны эшләтеп җибәргәннән соң бер-бер артлы Form1 формасына, Memo1 кырына,

Button1, Button2, Button3 төймәләренә һәм CheckBox1 гә чиртсәң килеп чыга торган нәтиҗәсе күрсәтелгән:



Рәс. 30. Кушымтаның эш нәтиҗәсе

4. ГАДӘТТӘН ТЫШ ОЧРАКЛАРНЫ ЭШКӘРТҮ

Күп эшләүчеләр Delphi да хәйран катлаулы программалар төзиләр. Катлаулы программа операцион система һәм аның кушымталары белән төрле яклап үзара тәэсир итешүне күздә тота. Бу тәэсир итешүнең теләсә кайсы ялгыш тәмамланырга мөмкин. Мисалларны күп китерергә була – нульгә бүлүдән башлап булмаган файлны ачуга кадәр. Гадәттә мондый очракларны әйләнеп үтү өчен программалаучыга күпсанлы тикшерүләр кертергә туры килә. Тик бер эшләүче дә аның программасы операцион система һәм башка кушымталар белән тәэсир итешкәндә килеп чыгарга мөмкин очракларны карап бетерә алмый. Нәкъ менә шундый алдан карап булмый торган вакыйгалар өчен гадәттән тыш очракларны структуралы эшкәртү уйлап табылган. Башта ул Windows NT белән эшләүчеләр өчен ясала, аннары Windows 9x операцион системаларында да кулланыла башлый. Хәзерге вакыттагы программа мохитләре Windows та кушымталар төзегәндә гадәттән тыш очракларны эшкәртүне башкаралар. Delphi мохите дә читтә калмады. Гадәттән тыш очракларны эшкәртү Delphi 1.0 дә үк кертелә, Delphi 2.0

дән башлап гадәттән тыш очраклар Win32 API ның бер өлеше булып тора.

4.1. Гадәттән тыш очрак төшенчәсе, аны Delphi чаралары белән эшкәртү

Гадәттән тыш очрак (исключение) дип без нииндидер алдан күреп булмый торган, программаның эш барышына тәэсир итә ала торган вакыйганы аңларбыз.

Мондый очракны эшкәрткәндә Delphi, гадәттәгечә, объектлар белән эшли. Delphi компиляторы өчен гадәттән тыш очрак – ул объект. Шундый специфик объект белән эшләү өчен Delphi да (төгәлрәк әйткәндә Object Pascal дә) түбәндәге конструкцияләр кертелә: *try .. except* һәм *try .. finally*.

try .. except конструкциясенең синтаксисы түбәндәгечә:

try
{башкарылучы код};
except
on Exception1 do
{1 нче хата килеп чыкканда башкарылучы код};
on Exception2 do
{2 нче хата килеп чыкканда башкарылучы код};
else
{югарыда каралмаган вакыйгаларны эшкәртүче код};
end;

Әгәр try бүлегендәге код башкарылганда гадәттән тыш очрак килеп чыкса, бу бүлекне башкару туктатыла һәм идарә except бүлегендә урнашкан кодка тапшырыла. Except бүлеге ике төрле юл белән кулланылырга мөмкин. Беренчедән, анда on нан башланган гадәттән тыш очракны эшкәртүчеләрдән башка теләсә нинди операторлар урнашырга мөмкин. Алар хата турында хәбәр операторлары булырга мөмкин, шулай ук система ресурсларын бушатучы командалар, башка операторлар һәм командалар булырга мөмкин. Икенчедән, except бүлеге гадәттән тыш очракларны эшкәртү өчен кулланыла. Бу очракта аңа бары тик гадәттән тыш очракны эшкәртүче операторлар гына керә ала. Әгәр эшкәртүчеләр арасында килеп чыккан гадәттән тыш очракка туры килүче эшкәртүче табылса, бу эшкәртүченең операторы башкарыла, гадәттән тыш очрак җимерелә һәм идарә on Exception do операторыннан соң урнашкан кодка

тапшырыла. Else ачкыч сүзеннән соң урнашкан бүлек except бүлегендә тасвирланмаган теләсә нинди гадәттән тыш очракларны эшкәртүөчен хезмәт итә. Бу бүлекнең булуы мәҗбүри түгел. Әгәр гадәттән тыш очракны эшкәрткәндә кирәкле эшкәртүче табылмаса, бу очракны эшкәртү система эшкәртүчесе тарафыннан башкарылачак.

Нульгә бүлү гадәттән тыш очрагын эшкәртү мисалын карыйк.

```
try
a:=10;
b:=0;
c:=a/b;
except
on EZeroDivide do MessageBox('Нульгә бүләргә ярамый!');
end;
```

Югарыда китерелгән мисалдан күренгәнчә, төрле очракларны эшкәртү өчен төрле операторлар хезмәт итә. Эшкәртүнең *on .. do* операторын тәфсилләбрәк карыйк. Бу оператор except бүлегендә урнаша һәм ике формада була ала.

on <гадәттән тыш очрак классы > do <оператор>;

яки

on <имя>: < гадәттән тыш очрак классы > do < гадәттән тыш очракның үзлекләрен кулланып була торган операторлар >

Бу оператор анда күрсәтелгән гадәттән тыш очраклар классын гына эшкәртә. Нәсел башы (база) классын күрсәткәндә, барлык аның токымнары булган гадәттән тыш очрак класслары шулай ук эшкәртеләчәк. Бөтен гадәттән тыш очракларны эшкәртү өчен аларның барысының база классы булган Exception га мөрәҗәгать итәргә мөмкин. Гадәттән тыш очракны эшкәрткәннән соң ул җимерелә. Операторның икенче формасы *on .. do* беренчесеннән бирелгән гадәттән тыш очракка вакытлыча исем биреп булу һәм гадәттән тыш очракның үзлекләренә мөрәҗәгать итеп булу белән аерыла. Гадәттән тыш очракның үзлекләренә *<исем>.<үзлек исеме>* конструкциясе ярдәмендә мөрәҗәгать итеп була. Киләсе кодны карыйк.

```
try
ScrollBar1.Max := ScrollBar1.Min - 1;
except on E: EInvalidOperation do MessageDlg( 'Гадәттән тыш очракка игътибар итмибез: ' + E.Message, mtInformation, [mbOK], O)
end;
```

Китерелгән мисалда без *EInvalidOperation* гадәттән тыш очрагына вакытлы Е исеме бирәбез. Аннары хәбәр тәрәзәсенә E.Message хата текстын чыгарабыз, ул Delphi тарафыннан килешү буенча чыгарыла (әгәр безнең хата эшкәртүче булмаган булса).

Искәрмә. Гадәттән тыш очракның вакытлы объектын юк итмәгез. Гадәттән тыш очракларны эшкәртүче үзе автомат рәвештә гадәттән тыш очрак объектларын юк итә, җимерә. Әгәр гадәттән тыш очракның объектын мөстәкыйль җимерсәгез, бу мөрәҗәгать итү, керү хатасына китерергә мөмкин.

Кайбер вакытта гадәттән тыш очракны үз коды белән эшкәрткәннән соң стандарт хата эшкәртүчене чакыру кирәк була. Мәсәлән, ниндидер хата килеп чыкканда кушымтаның кулланучыга ниндидер мәгълүматны хәбәр итүе, ә аннары идарәне стандарт хата эшкәртүчегә тапшыруы кирәк. Югарыда әйтелгәнчә, код тарафыннан гадәттән тыш очрак эшкәртелгәч, гадәттән тыш очрак юк ителә. Бу гадәттән тыш очракны яңадан чакыру өчен, гадәттән тыш очраклар регенерациясен (тергезүен) кулланырга мөмкин. Регенерация өчен *raise* командасы хезмәт итә. Мисал карыйк:

```
try
{операторлар}
except on < гадәттән тыш очрак классы> do
begin { гадәттән тыш очракны эшкәртү операторлары }
 raise; // Гадәттән тыш очракны регенерацияләү
end;
end;
```

Программалаучы язган гадәттән тыш очракны эшкәртү операторлары башкарылгач, *raise* командасы үтәлә, ул яңадан мәҗбүри гадәттән тыш очракны чакыра, аннан соң идарә гадәттән тыш очракларны стандарт эшкәртүчегә тапшырыла. Гадәттән тыш очрак кушымта кодындагы барлык *try* блоклары аша уңышлы үтсә, HandleException ысулы чакырыла. Ул диалог хата тәрәзәсен күрсәтә. Бу ысулны түбәндәгечә чакырып була:

```
try
 { операторлар }
except Application.HandleException(Self);
end;
```

Киләсе *try .. finally* конструкциясе *finally* бүлегендәге кодны ниндидер сәбәпләр аркасында *try* бүлегендә килеп чыга ала торган

гадәттән тыш очраклардан саклау өчен хезмәт итә. Бу конструкциянең синтаксисы:

try

{гадәттән тыш очрак китереп чыгара ала торган операторлар };

finally

{теләсә нинди очракта үтәлә торган сакланган операторлар };

end;

Димәк, *finally* ачкыч сүзеннән соң урнашкан операторлар теләсә кайсы очракта, гадәттән тыш очрак килеп чыкса да, чыкмаса да үтәләчәк. *try* бүлегендә гадәттән тыш очрак генерацияләнсә, идарә шунда ук *finally* бүлегенә тапшырыла. Шулай ук, *try* бүлегендә гадәттән тыш очрак булмаса, *finally* бүлеге үтәләчәк. Хәтта *finally* бүлегендә хата килеп чыкса да, бу бүлек операторларын башкару ахырга кадәр дәвам итәчәк. *try .. finally* конструкциясендә гадәттән тыш очракларны эшкәртү башкарылмый, ул башлыча хәтернең ресурсларын бушату, кирәкмәгән файлларны ябу һәм башка ресурс бушату операцияләре өчен кулланыла. Шул рәвешле бу конструкция файллар, хәтер, Windows ресурслары һәм объектлар белән операцияләр өчен кирәк. Гадәттән тыш очракны эшкәртү кодын *try .. except .. end* һәм *try .. finally .. end* блокларына бүләргә мөмкин. Бу блоклар бер-берсенә куелган була ала.

Delphi да кушымталар эшләгәндә еш кына программалаучыга гадәттән тыш очракларны эшкәртү түгел, ә бәлки хата китереп чыгаручы кирәкмәгән гамәлләрне туктатырга гына кирәк була. Моның өчен сүзсез дип аталучы гадәттән тыш очраклар (silent exceptions) кулланыла. Сүзсез гадәттән тыш очраклар стандарт EAbort гадәттән тыш очрагының токымнары булып тора. Килешү буенча VCL Delphi хаталар эшкәртүчесе экранга EAbort токымнарыннан кала барлык гадәттән тыш очраклар өчен хата диалог тәрәзәсе чыгара.

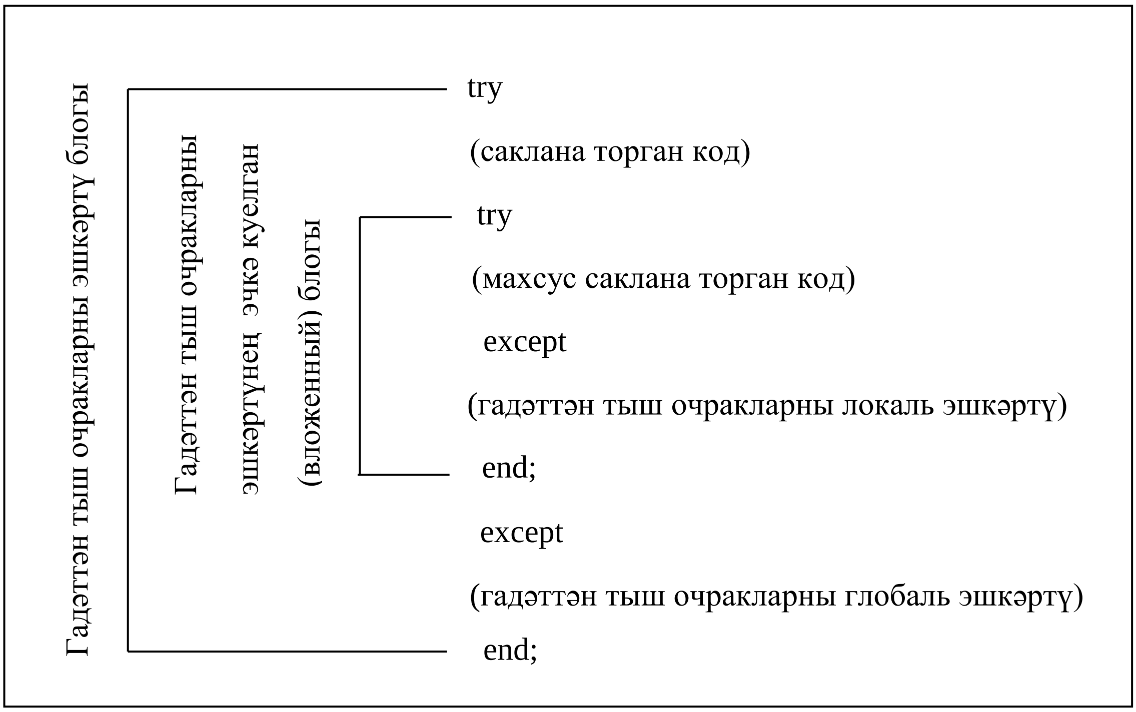
Искәрмә. Консоль кушымталар төзегәндә хата турында мәгълүмат эшкәртелмәгән EAbort гадәттән тыш очраклары өчен дә чыгарыла.

Сүзсез гадәттән тыш очрак генерацияләү өчен Abort процедурасын чакырырга мөмкин. Ул автомат рәвештә EAbort гадәттән тыш очрагын китереп чыгарачак. EAbort агымдагы операцияне экранга хата турында мәгълүмат чыгармыйча өзәчәк. Мисал карап үтик. Формада буш исемлек (ListBox1) һәм төймә

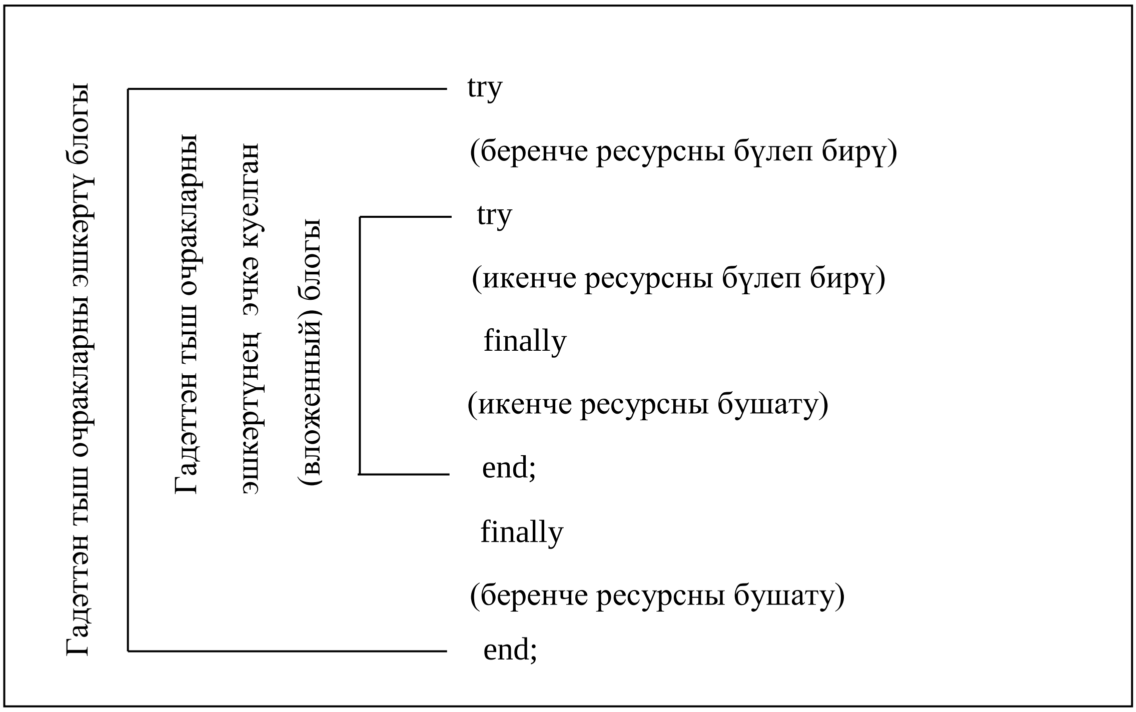
(Button1) булсын. Төймәнең onclick вакыйгасын эшкәртүчесенә түбәндәге кодны языйк:

```
procedure TForm1.Button1Click(Sender: TObject);
 var I: Integer;
begin
for I := 1 to 10 do    // 10 тапкыр цикл
 begin
 ListBox1.Items.Add(IntToStr(I));
                    { номерны исемлеккә өстибез }
if I = 7 then Abort;
                    { җиденче номерны өстәгәннән соң номерларны
           исемлеккә өстәүне туктатабыз }
end;
end;
```

Программа эше нәтиҗәсендә, Button1 төймәсенә баскач, исемлеккә 1 дән 7 гә кадәр номерлы җиде юл өстәләчәк.



Рәс. 31. а) Гадәттән тыш очракларны эшкәртүдә эчкә куелган (вложенный) блок



Рәс. 31. б) Кодны саклау конструкциясендә эчкә куелган (вложенный) блок

4.2. RTL-гадәттән тыш очракларын эшкәртү. Гадәттән тыш очракларның иерархиясе

RTL (real time library)-гадәттән тыш очраклар Delphi ның sysutils модулендә билгеләнгән һәм Exception база классының токымнары булып тора. Аларның берничә тибы бар.

RTL гадәттән тыш очраклары типлары

Хата тибы	Сәбәбе	Мәгънәсе
Кертү/чыгару	Файлга яки кертү/чыгару җайланмасына мөрәҗәгать итүдә, керүдә хата	Кертү/чыгару гадәттән тыш очракларының күбесе файлга мөрәҗәгать иткәндә Windows кайтара торган хата коды белән бәйле
Күч	Динамик хәтерне куллану хатасы	Күч хаталары хәтер җитмәгәндә яки кушымтада күч булмаган хәтер өлкәсенә күрсәткеч булганда килеп чыга
Бөтен саннар	Бөтен типтагы	Мондый хаталар була ала:

белән математик гамәлләр	аңлатма белән ялгыш эшләү	нульгә бүлү, тулып ташу, диапазон чикләреннән чыгу һ.б.
Йөзмә нокталы саннар белән математик гамәлләр	Реаль типтагы аңлатма белән ялгыш эшләү	Реаль саннар белән эшләгәндәге хаталар математик сопроцессордан яки программа эмуляторыннан булырга мөмкин. Хаталарга дөрес булмаган инструкцияләр, нульгә бүлү, тулып ташу һ.б. керә.
as операциясе	as операциясе ярдәмендә класслар белән ялгыш эш	Объектлар ярашлы объектлар белән генә эшли ала
Үзгәртү	Типларны ялгыш үзгәртү	Типларны үзгәртү функцияләре (IntToStr, StrToInt һ.б.)үзгәртү мөмкин булмаганда бу хатаны генерацияли
Аппарат	Система шартлары	Аппарат хаталары процессор яки кулланучы хатаны генерацияләгәнен күрсәтә: эш мөмкинлеге, керү хатасы, стәкнең тулып ташуы һ.б.
Variant тибы	Variant тибын ялгыш куллану	Хата Variant тибын кулланып булмаган аңлатмаларда чыга

Гадәттән тыш очраклар класслары иерархиясен җентекләбрәк карыйк:

Exception	гадәттән тыш очракларның база классы
EAbort	хисаплауны алдан уйлап, юри өзү
EAbstractError	абстракт методны чакырырга тырышу
EAccessViolation	хәтергә мөрәҗәгать итү, керү хатасы
EArrayError	массивлар белән эшләү хатасы

EAssertionFailed	хаклыкны тикшергәндәге хата
EBitsError	TBits буль кыйммәтләре массивына мөрәҗәгать итү, керү хатасы
ECacheError	кәш төзү хатасы
EComponentError	компонентны регистрацияләү яки исемен алыштыру хатасы
EControlC	консоль кушымта башкарылганда кулланучы тарафыннан <Ctrl>+<C> төймәләре басылу
EConvertError	юлларны (объектларны) үзгәртү хатасы
EDatabaseError	мәгълүмат базалары белән эш хатасы ошибка
EDBClient	клиент мәгълүматын җыюда хата
EReconcileError	TClientDataset компонентының мәгълүматларын яңартуда хата
EDBEngineError	BDE да хата
ENoResultSet	select сыз гына сорауны (запрос) ачарга тырышканда TQuery компоненты тарафыннан генерацияләнә
EDateTimeError	дата һәм вакыт кертү хатасы
EDimensionMapError	чишелешләр кубында мәгълүмат форматы хатасы
EDimIndexError	чишелешләр кубының үлчәмен билгеләүдә ялгыш индекс
EExternalException	билгесез гадәттән тыш очрак
EInOutError	файлга кертү/чыгару хатасы
EIntError	математик операцияләр өчен гадәттән тыш очракларның база классы
EDivByZero	нульгә бүлү хатасы
ERangeError	рөхсәт ителгән диапазоннан читтәге кыйммәт яки индекс
EIntOverflow	тулып ташу
EIntfCastError	as типларын интерфейска ялгыш үзгәртү
EInvalidGraphic	танып булмый торган график файл
EInvalidGraphicOperation	графика белән операцияләрдә хата
EInvalidGridOperation	челтәр (Grid) белән эшләгәндәге хата
EInvalidOperation	компонент белән ялгыш гамәл

	(операция)
EInvalidPointer	күрсәткеч белән ялгыш гамәл (операция) хатасы
EListError	исемлек белән эшләгәндә хата
ELowCapacityError	чишелешләр кубына хәтер бүлү хатасы
EMathError	йөзмә нокта белән гамәлләрнең (операцияләр) гадәттән тыш очракларының база классы
EInvalidArgument	математик функциягә мөрәҗәгать иткәндә параметрның ярамаган кыйммәте
EInvalidOp	билгесез операция
EOverflow	тулып ташу хатасы
EUnderflow	мәгънәле разрядларны югалту
EZeroDivide	нульгә бүлү хатасы
EMCIDeviceError	MCI (Media Control Interface) драйверы аша җайланмалар белән эш мөмкинлеге хатасы
EMenuError	сайлак элементлары белән эш хатасы
EOleCtrlError	кушымтаны ActiveX элементы белән бәйләгәндәге хата
EOleError	OLE түбән кат (дәрәҗә) хатасы
EOleSysError	OLE IDispatch интерфейсы хатасы
EOleException	үзлек яки ысул (метод) белән бәйле OLE хатасы
EOutlineError	toutline белән эш хатасы
EOutOfMemory	хәтерне бүлеп бирү хатасы
EOutOfResources	Windows эшкәртүчесе ясау хатасы
EPackageError	Пакетны йөкләү яки куллану вакытында генерацияләнүче гадәттән тыш очрак
EParserError	форма тасвирламасы текстын икешәрле форматка күчерү хатасы
EPrinter	бастыру хатасы
EPrivelege	привилегияләр җитмәү сәбәпле процессор инструкциясен үтәү хатасы
EPropReadOnly	OLE кыйммәт ярдәмендә укырга гына дигән үзлекне язу хатасы

EPropWriteOnly	OLE кыйммәт ярдәмендә язарга гына мөмкин үзлекне уку хатасы
EPropertyError	үзлекнең кыйммәтен биргәндәге хата
ERegistryException	Windows реестры белән эш хатасы
EReportError	сервер тибын билгеләү хатасы (TReport компоненты мәгълүмат базасы белән тоташа алмый)
EResNotFound	кушымтаны төзегән вакытта ресурслар файлын (*.DFM яки *.RES) йөкләү хатасы
EStackOverflow	стэк тулып ташу
EStreamError	агым гадәттән тыш очракларының база классы
EFCreateError	файл төзү хатасы
EFOpenError	файл ачу хатасы
EFilerError	файл агымнары гадәттән тыш очракларының база классы
EReadError	бирелгән сандагы байтларны уку хатасы
EWriteError	бирелгән сандагы байтларны язу хатасы
EclassNotFound	компонентны кушымта белән бәйләү хатасы
EInvalidImage	ресурслар файлын уку хатасы
EmethodNotFound	ысул (метод) табылмау
EStringListError	исемлек тәрәзәсенә керү хатасы
EThread	күп агымлы кушымта хатасы
ETreeViewError	TTreeview белән эш вакытында индекс хатасы
EVariantError	variant тибындагы бирелмәләр (мәгълүмат, данные) белән эш хатасы
EWin32Error	Windows ның эчке хатасы

Шулай итеп, Exception классы Delphi ның барлык гадәттән тыш очраклары өчен база классы булып тора. Югарыда тасвирланган барлык класслар Exception классының туры яки турыдан-туры булмаган токымнары булып тора. Гадәттән тыш очракларның яңа классын төзегәндә Exception классын баба класс итеп кулланырга кирәк. Шулай булганда гына Delphi яңа классны гадәттән тыш очрак

буларак таныячак һәм эшкәртәчәк. Үз чиратында Exception классы TObject база классының туры токымы булып тора һәм аның барлык функцияләрен мирас итеп ала. Башка класслардан аермалы буларак, гадәттән тыш очрак класслары Т хәрефеннән түгел, Е хәрефеннән башлана. Яңа классларны төзегәндә, аларга үзең теләгәнчә исем биреп була, Е хәрефеннән башланмаса да ярый (тик бу программалауның начар стиле булып тора).

4.3. Үз гадәттән тыш очракларыңны төзү

Үз гадәттән тыш очракларыңны төзү өчен гадәттән тыш очракның объекты тибын ничек билгеләргә һәм мондый очракны ничек чакырырга икәнен белергә кирәк.

Гадәттән тыш очрак Delphi объекты булганга күрә, андый очракның яңа тибын билгеләү катлаулы түгел, объектның яңа тибын билгеләү кебек гади эш. Теоретик яктан караганда теләсә кайсы объектны гадәттән тыш очрак объекты кебек чакырып була, тик гадәттән тыш очракларны стандарт эшкәртүчеләр нәсел башы булып Exception яки Exception токымнары торган объектлар белән генә эшлиләр. Үз гадәттән тыш очрагыңны төзү мисалы итеп чираттагы билгеләмәне карыйк:

type
EMyException = class(Exception);

Бу вакытта EMyException гадәттән тыш очрагы чакырылса, ләкин аның өчен эшкәртүче язылмаса, Exception өчен булган стандарт эшкәртүче чакырылачак. Exception өчен булган стандарт эшкәртүче чакырылган гадәттән тыш очракның исемен күрсәткәнгә күрә, яңа гадәттән тыш очракның исемен күреп булачак.

Төзелгән гадәттән тыш очракны чакыру өчен raise командасы кулланыла. Мисал өчен кулланучы тарафыннан кертелгән парольне тикшерү мәсьәләсен карыйк:

type
EPasswordInvalid = class (Exception);

EpasswordInvalid гадәттән тыш очрагының яңа тибын билгеләгәннән соң бу гадәттән тыш очракны программаның теләсә кайсы урынында чакырып була:

if Password <> CorrectPassword then raise EPasswordInvalidCreate ('Введен неправильный пароль');

Төзелгән гадәттән тыш очракны аның исеме буенча чакыралар.

5. ИНТЕРФЕЙСЛАР ҺӘМ АЛАРНЫ КЛАССЛАР ТАРАФЫННАН БЕРГӘ КУЛЛАНУ

Delphiның interface дигән ачкыч сүзе кушымталарда интерфейсларны төзергә һәм кулланырга мөмкинчелек бирә. Интерфейслар VCL да мирас итеп алу моделен киңәйтү өчен хезмәт итәләр. Алар бер класска берничә интерфейска карарга, шулай ук төрле база класслары токымнары булган берничә класска бер интерфейсны кулланырга мөмкинчелек бирәләр. Интерфейслар агым кебек гамәлләр (операцияләр) җыелмалары күп сандагы объектлар тарафыннан кулланылганда файдалы.

Шул рәвешле, интерфейслар – төрле объектлар арасында бәйләнешләрне көйләү чаралары.

Интерфейслар компонентлы объект моделе (COM) яки CORBA технологияләре өчен нигез булып тора.

Интерфейслар үз эчләрендә абстракт ысуллар (методлар) һәм аларның функциональлегенең ачык билгеләмәләре генә булган классларга охшаш. Интерфейсның ысулын (методын) билгеләү үз эченә параметрларны, параметрларның типларын, кайтарыла торган тип һәм көтелүче тотышны ала. Интерфейс ысуллары (методлары) семантик яки логик яктан интерфейсның максатын чагылдыру белән бәйле. Интерфейслар турында килешү бар, аның буенча һәр интерфейска исем ул башкарачак мәсьәлә белән бәйле бирелергә тиеш. Мәсәлән, IMalloc интерфейсы хәтерне бүлеп бирү, бушату һәм хәтер белән идарә итү өчен билгеләнгән. Шул рәвешле IPersist интерфейсы токымнары өчен база интерфейсы буларак кулланыла ала. Токымнарының һәркайсы объектлар торышын йөкләү һәм хәтердә, агымда яки файлда саклау ысулларының (методларының) специфик прототипларын билгели. Интерфейсны игълан итүнең гади мисалын китерик.

```
Type
 IEdit = interface procedure Copy;
 stdcall;
procedure Cut;
stdcall;
 procedure Paste;
stdcall;
function Undo: Boolean;
stdcall;
```

end;

Интерфейс ярдәмендә интерфейс экземпляры төзергә ярамый. Интерфейс экземпляры алу өчен аны бу интерфейсны үзенә алучы класста игълан итәргә кирәк. Шулай итеп, кирәкле интерфейс үзенең баба класслары исемлегендә булган классны билгеләргә кирәк.

```
TEditor = class(TInterfacedObject, IEdit)
 procedure Copy;
 stdcall;
procedure Cut;
 stdcall;
procedure Paste;
 stdcall;
function Undo: Boolean;
stdcall;
end;
```

Югарыда әйтелгәнчә, интерфейсларны куллану берничә класска бер интерфейсны кулланырга мөмкинчелек бирә, бер база баба (ана) классы булуын таләп итми. Интерфейс – ул яшәү вакыты белән идарә итеп була торган тип, ягъни ул инициализация вакытында автомат рәвештә nil кыйммәте ала, аның сылтамалар санагычы бар һәм ул үзенең күренүчәнлек өлкәсе чикләреннән чыкканда автомат рәвештә юк ителә.

5.1. IUnknown интерфейсы

Барлык классларның бабасы булып TObject классы торган кебек, бөтен интерфейслар да IUnknown интерфейсының турыдан туры яки турыдан туры булмаган варислары. Бу төп интерфейс System модулендә түбәндәгечә тасвирланган

```
type
IUnknown = interface ['{00000000-0000-0000-C000-000000000046}']
 function QueryInterface(const IID: TGUID; out Obj): Integer;
 stdcall;
 function _AddRef: Integer;
stdcall;
 function _Release: Integer;
stdcall;
```

end;

Интерфейсны тасвирлау синтаксисы класс тасвирлауныкына охшаган. Төп аерма шунда – интерфейс глобаль уникаль идентификатор (Global Unique Identifier, GUID) белән бәйле була ала.

GUID – ул 128 разрядлы бөтен сан, интерфейсларга уникаль идентификация үткәрү өчен кулланыла. 128 разрядта күп саннар кодлаштырып булганга күрә GUID интерфейсның глобаль уникаль идентификаторын гарантияли. Ягъни ике компьютерда GUIDның тәңгәл килүе практикада мөмкин түгел. GUIDны генерацияләү компьютерның аппарат өлкәсенә нигезләнә (процессор ешлыгы, челтәр картасы номеры һ.б.). APIның CocreateGUID() функциясе ярдәмендә реализацияләнә ала торган алгоритм эше нәтиҗәсендә TGUID тибындагы язма барлыкка килә. Бу язманы түбәндәге форматтагы юл рәвешендә билгеләп була:

'{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}'

COMны караганда без һәр интерфейс яки COM классының үз GUIDы бар икәнен күрербез. Интерфейслар өчен – ул интерфейс идентификаторы (Interface ID, IID), класслар өчен – класс идентификаторы (Class ID, CLSID).

Delphi мохитендә яңа GUID төзү өчен код редакторы тәрәзәсендә төймәләрнең <Ctrl>+<Shift>+<G> комбинациясен басу җитә.

Iunknown интерфейсы өч ысул (метод) белән эшли, алар барлык интерфейслар тарафыннан мирас итеп алына:

- QueryInterface() – бирелгән интерфейс белән эшләп буламы икәнлеге турында сорау (запрос) төзегәндә кулланыла, әгәр дә җавап уңай икән, ысул (метод) аңа күрсәткечне кайтара. Мисал өчен ниндидер object объекты бар дип карыйк, ул берничә интерфейс: interface1, interface2 белән эшли ала. Object объектының interface2 интерфейсына күрсәткечне алу өчен Interface2.Query Interface () ысулын (методын) чакырырга кирәк;

- _AddRef () – бирелгән интерфейска күрсәткеч алынгач, һәм бу күрсәткеч белән эшләргә кирәк булганда кулланыла. _AddRef() ысулы (методы) мәҗбүри рәвештә _Release () ысулын (методын) чакыру белән тәмамланырга тиеш;

- _Release () – интерфейс белән эшне төгәлләү өчен кулланыла.

Интерфейслар COM яки CORBA кебек бүленгән объект модельләренең фундаменталь элементлары булып тора.

5.2. TInterfacedObject классы

Delphi VCLда TInterfacedObject классы билгеләнгән, ул интерфейс объектлары өчен база классы булып тора. Бу класс Delphi system модулендә түбәндәгечә билгеләнгән

type

TInterfacedObject = class(TObject, IUnknown)

private FRefCount: Integer;

protected function QueryInterface(const IID: TGUID; out Obj): Integer;

stdcall;

function _AddRef: Integer;

stdcall; function _Release: Integer;

stdcall;

public property RefCount: Integer read FRefCount;

end;

Күренгәнчә, бу классның баба класслары, нәсел башы булып TObject классы һәм IUnknown интерфейсы тора. TInterfacedObject классы ярдәмендә интерфейс белән эшли ала торган классларны җиңел төзеп була. Мәсәлән,

type

TMyObjInterfaced = class(TInterfacedObject, IPaint)

end;

Без яңа TMyObjInterfaced классын билгелибез, ул TInterfacedObject классының туры токымы булып тора һәм IPaint интерфейсы белән эшли ала.

5.3. as операторын куллану

Интерфейслар белән эшләүче объектлар as операторын интерфейсны динамик тоташтыру өчен куллана алалар. Мисал өчен,

procedurePaintObjecta(P: TInterfacedObject)

var X: IPaint;

begin X := P as IPaint;

{операторы}

end;

Бу мисалда P үзгәрешлесенең тибы TInterfacedObject. Бу үзгәрешле IPaint интерфейсына сылтама буларак x үзгәрешлесенә билгеләнә ала. Мондый билгеләү өчен компилятор P үзгәрешлесенең IUnknown интерфейсына бәйле QueryInterface ысулын (методын)

чакырырга код генерацияли. Мондый билгеләү P бу интерфейс белән эшләмәгән очракта да мөмкин. Ягъни компилятор бу билгеләү вакытында хата күрсәтмәячәк.

Югарыдагы мисал үтәлгәндә яки

X:= P as IPaint;

үзләштерүе уңышлы башкарыла, яки гадәттән тыш очрак генерацияләнә.

as операторын кулланганда түбәндәге таләпләр үтәлергә тиеш:

- интерфейсны игълан иткәндә баба (ана) буларак IUnknown интерфейсын күрсәтергә. Чөнки бу очракта гына as операторын кулланып була;

- интерфейс өчен as операторы кулланса, бу интерфейсның үз IID булырга тиеш. Яңа IID төзү өчен код редакторында төймәләрнең <Ctrl>+<Shift>+<G> комбинациясен сайларга кирәк.

5.4. implements ачкыч сүзен куллану

VCL Delphi да күп классларның кайбер үзлекләре булып объектлар тора. Моннан тыш класс үзлекләре сыйфатында интерфейсларны кулланырга мөмкин. Үзлек интерфейс тибында булса, бу интерфейс объектка тапшыра торган ысулларны (методларны) билгеләү өчен implements ачкыч сүзен кулланырга мөмкин. Килешү буенча implements ачкыч сүзе интерфейсның барлык ысулларын (методларын) тапшыра. Шуңа да карамастан, интерфейсның объектка ташырылучы ысуллар (методлар) исемлеген мөстәкыйль билгеләп була.

Листингта implements ачкыч сүзен төс адаптеры объекты төзегәндә куллану китерелгән. Ул RGB төсенең сигез битлы кыйммәтен үзгәртүөчен кулланыла.

```
unit cadapt;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
IRGBSbit = interface ['{Id76360a-f4f5-lldl-87d4-00c04fbl7199}']
function Red: Byte;
function Green: Byte;
function Blue: Byte;
```

end;
IColorRef = interface [41d76360b-f4f5-lldl-87d4-00c04fbl7199}']
function Color: Integer;
end;
TRGBSColorRefAdapter = class(TInterfacedObject, IRGBSbit, IColorRef)
private
FRGBSbit: IRGBSbit; FPalRelative: Boolean;
public
constructor Create(rgb: IRGBSbit);
property RGBSIntf: IRGBSbit
read FRGBSbit
implements IRGBSbit;
property PalRelative: Boolean
read FPalRelative
write FPalRelative;
function Color: Integer;
end;
implementation
constructor TRGBSColorRefAdapter.Create(rgb: IRGBSbit);
begin FRGBSbit := rgb;
end;
function TRGBSColorRefAdapter.Color: Integer;
begin if FPalRelative then Result := PaletteRGB(RGBSIntf.Red, RGBSIntf.Green, RGBSIntf.Blue) else Result := RGB(RGBSIntf.Red, RGBSIntf.Green, RGBSIntf.Blue);
end;
end.

5.5. Интерфейсларны бүленгән кушымталарда куллану

Интерфейслар COM һәм CORBA объектларның бүленгән компонентлары модельләренең фундаменталь элементлары булып тора. Delphi бу технологияләрне база класслары белән тәэмин итә. Алар TInterfacedObject объекты мөмкинчелекләрен киңәйтә.

COM класслары класс фабрикалары һәм класс идентификаторлары куллану мөмкинлеген өстиләр (CLSID). Класс фабрикалары CLSID аша класс экземплярлары төзү өчен җавап бирәләр. Үз чиратында CLSID COM классларын регистрацияләү һәм

алар белән манипуляцияләр өчен кулланыла. Класс фабрикалары да, класс идентификаторлары да булган COM класслары CoClasses дип атала. Интерфейсларның яңа версияләре белән эшләгәндә CoClassesларның QueryInterface ка караганда өстенлеге бар. Иске интерфейсларның яңа версияләре программа-клиентлар өчен автомат рәвештә ачык, эшкә яраклы була. COM кушымталарында программалаучы кушымтаның эшен яхшырту өчен кодның клиент өлешен үзгәртмичә интерфейс кодына үзгәреш кертә ала.

Икенче бүленгән технология CORBA (Объектларга таләп брокеры гомуми архитектурасы) дип атала. Ул төрле аппарат яки программа платформалары белән үзара тәэсир итешү өчен кушымталар төзергә мөмкинчелек бирә. Мәсәлән, Windows операцион системасында эшләүче CORBA кушымтасы UNIX операцион системасының кушымталар серверы белән дә җиңел эшләячәк.

Әдәбият

1. Немцова Т.И., Голова С.Ю., Абратов И.В. Программирование на языке высокого уровня. Программирование на языке Object Pascal: учебеное пособие / под ред. Л.Г.Гагариной. – М.: ИД «ФОРУМ»: ИНФРА-М. 2015. – 496 с.
2. Фаронов В. В. Delphi : программирование на языке высокого уровня : учеб. для студ. / Санкт-Петербург: Питер, 2005 . – 640 с.
3. Бобровский С. И. Delphi 7 : учеб. курс/ Санкт-Петербург: ПИТЕР, 2005 . – 736 с.
4. Павловская Т. А. С/С++. Программирование на языке высокого уровня : учеб. для студ. вузов, обуч. по напр. «Информатика и вычислительная техника»/ Санкт-Петербург: ПИТЕР, 2005 . – 461 с.
5. Иванова Г. С. Объектно-ориентированное программирование : учеб. для студ. / Москва: Изд-во МГТУ им. Н. Э. Баумана, 2003 . – 368 с.
6. Костюкова Н. И. Язык Си и особенности работы с ним : учеб. пособие/ Москва: Интернет-Университет Информационных Технологий: БИНОМ, 2006 . – 207 с.
7. Седжвик, Роберт, Фундаментальные алгоритмы на С++ / Санкт-Петербург : ДиаСофтЮП. – 2002. – 688 с.
8. Павловская Т.А. С/С++. Программирование на языке высокого уровня. / Т.А. Павловская. – СПб.: Питер, 2002. – 464 с.: ил.
9. Павловская Т.А, Щупак Ю. С/С++. Объектно-ориентированное программирование: Практикум. – СПб.: Питер, 2004 – 265 с.
10. Программирование на языке Delphi [Электронный ресурс] URL: http://rsdn.ru/?article/Delphi/Delphi_7_07.xml (дата обращения 12.05.2016)