Основные категории событий

- **События, обусловленные действиями пользователя** (пользовательские - *user events*);

перемещение мыши, нажатие клавиши...

- **Программно-управляемые события** (обычные);

создание или разрушение формы,...

- **Главная часть** программного кода приложений - это **процедуры обработки пользовательских событий**.
  Чаще всего это обработчики событий **мыши** и **клавиатуры**.
- **К обычным событиям** относятся события **активизации, завершения, события изменения состояния отдельных компонентов,** которые являются косвенным результатом действия пользователя.

Программно-управляемые события для форм (TForm)

OnCreate

- Событие происходит **один раз** за все время существования формы.
- При запуске приложения Delphi создает формы с помощью метода **Create**. При этом последовательно вызываются обработчики :
  - **OnCreate**
  - **OnShow**
  - **OnActivate**
  - **OnPaint**

В процедуру-обработчик **нельзя** включать **явных ссылок на саму форму**.

Пример. Следующая процедура случайным образом задает размеры создаваемой формы:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
Randomize; {инициализация датчика случайных чисел}
Height:= Random(500);
Width := Random(600); {заданы случайные размеры}
end;
```

OnClose

Обработчик вызывается **только** при использовании **кнопки закрытия** или **вызова метода Close**

и может изменить стандартное поведение формы при закрытии.

Особое поведение формы при закрытии

обеспечивает переменная **Action**

caNone

запрет на закрытие формы.

caHide

форма становится невидимой,
но продолжает существовать.

caFree

форма закрывается с высвобождением памяти.

caMinimize

форма минимизируется.

Процедура, которая закрывает форму только при нажатии кнопки "**Yes**" в окне диалога:

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
{MessageDlg используется как функция, дающая значение нажатой кнопки}
if MessageDlg('Закрыть окно ?', mtConfirmation,[mbYes, mbNo], 0) = mrYes then Action := caFree
{закрытие формы}
else Action := caNone; {запрет на закрытие}
end;
```

OnCloseQuery
Обработчик вызывается
только при использовании **кнопки закрытия** или **вызова метода Close** и
может изменить стандартное поведение формы при закрытии,
используя **специальный параметр CanClose**.
По умолчанию этот параметр имеет значение **True**.

Если в процедуре-обработчике события **OnCloseQuery** установить для параметра **CanClose** значение **False**, то форма останется открытой.

Пример. Форма является специальной и предназначена для ввода пароля.
Форма не должна закрываться до тех пор, пока пользователь не наберет какой-нибудь текст в строке ввода пароля.

В проекте должны присутствовать:

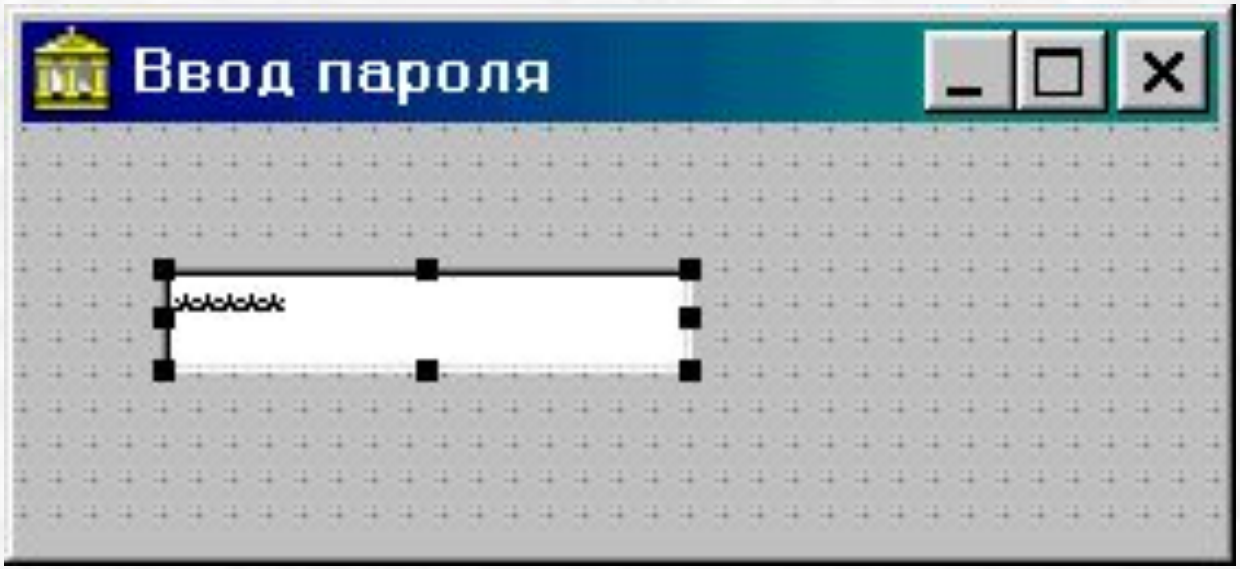
- **Форма** (свойство **Name - PasswordDlg**);

На форме:

- **Строка ввода** (свойство **Name - Password**)

(кроме того, могут быть кнопки типа OK, Cancel, Yes, No….)

Для строки ввода, используемой для указания пароля, необходимо установить свойство **PasswordChar - ***
(набираемые символы будут отображаться *)



```
procedure TPasswordDlg.FormCloseQuery(Sender:
TObject;  var CanClose: Boolean);
begin
if   Password.Text  =  ''
{если  строка ввода пустая, то ...}
then
   begin  {запретить закрытие}
   CanClose := False ;
{поместить фокус в строку ввода,
 в принципе, необязательно}
   ActiveControl := Password ;
   end;
end;
```

Пользовательские события

Основные группы пользовательских событий:

события мыши;

операции Drag&Drop

(перетащить и бросить);

события клавиатуры.

События мыши.

Действия пользователя с мышью.

Приложение может реагировать на следующие действия с мышью:

- Нажатие кнопки мыши - **MouseDown**;
- Отпускание кнопки мыши - **MouseUp**;
- Перемещение мыши - **MouseMove**;
- Щелчок - **Click**;
- Двойной щелчок - **DblClick**.

Основные типы обработчиков:

- **OnMouseDown**, **OnMouseUp**:

  TMouseEvent = **procedure** (Sender : TObject;
  Button : TMouseButton;
  Shift : TShiftState; X, Y : Integer) **of object**;

- **OnMouseMove**:

  TMouseMoveEvent = **procedure** (Sender :
  TObject; Shift : TShiftState; X, Y : Integer)
  of object;

- **OnClick**, **OnDblClick**:

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

Параметры процедур:

Sender объект, на который воздействует мышью пользователь.

Button нажатая кнопка мыши:
mbLeft, mbMiddle, mbRight

type TMouseButton = (mbLeft, mbRight, mbMiddle)

Shift состояние клавиш **Alt, Ctrl, Shift**:

type TShiftState = **set of** (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble)

X, Y координаты точки на экране, где произошло событие.

События клавиатуры.

Событие клавиатуры генерируется, как только будет **нажата** или **отпущена некоторая клавиша**. В этом случае **Windows** направляет приложению **соответствующее сообщение**.

Windows **самостоятельно** обрабатывает **большую часть** нажатий клавиш:

- **[Alt] + клавиша**

  для **вызова команд меню**,

- клавиши, используемые **для ввода или редактирования текста в TEdit, TMemo**.

  Их нажатие обрабатывать **не нужно**.

Куда направляются сообщения о событиях клавиатуры.

- Пользователь **производит ввод** с **клавиатуры**.
- **Сообщение о событии** получает **элемент управления**, имеющий в данный момент **фокус ввода**.
- **Событие** обрабатывается **процедурой-обработчиком именно этого элемента управления.**

- Можно организовать обработку **событий клавиатуры** самой **формой:**

перехват событий формой.

Для этого достаточно установить (**для формы!**) значение свойства:

KeyPreview = True.

В этом случае **сначала будут работать обработчики событий клавиатуры формы**,
а лишь **затем элемента управления**, имеющего **фокус ввода**.

- Это позволяет **контролировать обработку** нажатий клавиш.
- **Исключение составляют клавиши**:
  [Tab],
  [Backspace],
  клавиши со стрелками и т.п.,
  использующиеся в компонентах ввода и редактирования.

Обработчики событий клавиатуры

- **При нажатии клавиши вызываются два обработчика событий:**

OnKeyDown

OnKeyUp.

- Они **взаимосвязаны** и всегда **вызываются попарно**, так как всякая нажатая клавиша рано или поздно отпускается.
- Кроме того, для **каждой нажатой клавиши с символом ASCII** вызывается еще один **обработчик события**:

OnKeyPress.

OnKeyDown OnKeyUp

Обработчики событий позволяют **перехватывать** нажатие **сочетаний клавиш** (с использованием **[Shift], [Ctrl], [Alt]**), а также **функциональных клавиш (F1, ..)**.

- Пока пользователь держит клавишу **нажатой**, генерируются **повторяющиеся** события и вызывается обработчик **OnKeyDown**.
- После **отпускания** клавиши вызывается обработчик события **OnKeyUp**.

Оба обработчика имеют тип **TKeyEvent**:

TKeyEvent =

procedure (Sender: TObject; var Key: Word;

Shift: TShiftState) of object;

Параметры имеют следующий смысл:

Key - код нажатой клавиши.

Shift - состояние клавиш **Shift, Alt, Ctrl** и клавиш мыши.

Пример 1. При установке для Form1 свойства **KeyPreview = True**

следующая процедура обеспечит **многократное** изменение цвета формы при **нажатии и удерживании** клавиши **Shift**:

```
procedure TForm1.FormKeyDown(Sender: TObject; var
Key: Word; Shift: TShiftState);
begin
if Shift = [ssShift]
{если нажата клавиша Shift}
then Color := RGB(random(255), random(255),
random(255));
end;
```

Пример 2. При установке для **Form1** свойства **KeyPreview = True** следующие процедуры обеспечат **уменьшение** высоты формы при **нажатии** клавиши "пробел" и **увеличение** высоты при **отпускании** этой клавиши:

```
procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState);
begin
if chr(Key) = ' '
{если нажата клавиша "пробел"}
then Height :=100;
end;
```

```
procedure TForm1.FormKeyUp(Sender:
TObject; var Key: Word; Shift: TShiftState);
begin
if chr(Key) = ' '
{если отпущена клавиша "пробел"}
then Height :=200;
end;
```

Использование в сравнениях констант

При работе с параметром **Key** вместо кодов иногда гораздо удобнее использовать **специальные константы - виртуальные коды**. Особенно это целесообразно при использовании **специальных клавиш**, которым **нельзя** поставить в соответствие символы - **Enter, Esc, F1, F2 , ...**

Константа	16-тиричный код	Клавиша
VK_SHIFT	10	Shift
VK_CONTROL	11	Ctrl
VK_MENU	12	Alt
VK_ESCAPE	1B	Esc
VK_HOME	24	Home
VK_LEFT	25	←
VK_0	30	0
VK_J	4A	J
VK_NUMPAD0	60	0 на цифровом блоке клавиатуры
VK_F8	77	F8

Пример 3. При установке для **Form1** свойства **KeyPreview = True** процедура-обработчик события OnKeyDown обеспечит **перемещение формы в верхний левый угол** экрана при нажатии клавиши **[Home]:**

```
procedure TForm1.FormKeyDown(Sender: TObject;
var Key: Word; Shift: TShiftState);
begin
    if Key = VK_HOME
    then  begin
              Left :=0;
              Top :=0;
          end;
end;
```

Здесь **Left, Top -** свойства TForm, определяющие экранные координаты **верхнего левого угла формы**.

Пример 4. При нажатии сочетания клавиш **[Alt + F10]** появляется соответствующее сообщение:

```
procedure TForm1.FormKeyDown(Sender:
TObject; var Key: Word; Shift: TShiftState);
begin
if (Shift = ssAlt) and (Key = VK_F10)
then MessageDlg ('нажаты Alt+F10',
mtInformation, [mbOK], 0);
end;
```

OnKeyPress

Когда пользователь **нажимает клавишу с символом ASCII в дополнение** к ранее рассмотренным вызывается **еще один обработчик события**:
OnKeyPress,
имеющий тип **TKeyPressEvent**:

TKeyPressEvent = **procedure**
(Sender: TObject**; var** Key: Char) **of object;**

В отличие от типов обработчиков, приведенных выше:

Key - символ нажатой клавиши.

Клавиши, **не имеющие символьного значения**

([Alt], [Shift], [Ctrl], [F1], …),

не генерируют событий KeyPress.

В то же время, могут использоваться (в смысле обработки указанного события) клавиши **Esc, Enter, BackSpace**.

Пример 5. Приведенная ниже процедура представляет пример простейшей обработки события **KeyPress**.

При нажатии различных клавиш отслеживается, для каких из них генерируется событие **KeyPress** - в заголовок формы **добавляется символ, соответствующий нажатой клавише**.

При этом для некоторых клавиш (**Esc, Enter, BackSpace**) добавляется специфический символ -прямоугольник.

```
procedure TForm1.FormKeyPress(Sender:
TObject; var Key: Char);
begin
Form1.Caption := Form1.Caption  + Key;
end;
```

Обработка исключительных ситуаций

При работе программы возможно возникновение различных **нестандартных ситуаций** (деление на нуль, попытка работы с несуществующим файлом и т.п.) с нежелательными последствиями.

Такие ситуации называются **исключительными**.

Возможно программно отслеживать возникновение исключительных ситуаций – **обрабатывать исключительные ситуации**.

Непосредственные проверки утяжеляют алгоритм решения задачи.

- В Object Pascal можно вынести обработку исключительных ситуаций **в отдельные части** программы,
не смешивая ее с основным алгоритмом.
- В Object Pascal при возникновении исключительной ситуации создается специальный объект – **исключение**.

- Исключение несет в себе основную информацию об исключительной ситуации и должно быть специальным образом обработано.

Исключительные ситуации могут формироваться:

- **процессором,**
- **операционной системой,**
- **средой Delphi,**
- **самой программой** –
  - пользовательской частью,
  - стандартными подпрограммами.

Можно создавать **собственные исключения**, активизировать и обрабатывать их.

Структура исключения

Исключение с точки зрения языка Object Pascal представляет собой **класс**, являющийся **потомком класса исключений Exception**, который определяется в стандартном модуле **SysUtils** (системные вспомогательные средства).

Модуль SysUtils подключается автоматически.

```
type
Exception = class(TObject)
private
 FMessage: string;
 FHelpContext: integer;
public
 constructor Create(const Msg: string);
 constructor CreateFmt(const Msg:
 string;
                       const Args: array of const);
```

```
constructor CreateRes(Ident: integer);
constructor CreateResFmt(Ident: integer;
                         const Args: array of const);
constructor CreateHelp(const Msg:
string;
                       AHelpContext: integer);
constructor CreateFmtHelp(const Msg:
string;
                          const Args: array of const;
                                                    AHelpContext: integer);
```

```
constructor CreateResHelp(Ident:
  integer;
                          AHelpContext: integer);
constructor CreateResFmtHelp(Ident:
  integer;
                    const Args: array of const;
                          AHelpContext: integer);
  destructor Destroy;
  property HelpContext: integer;
  property Message: string;
```

Свойства класса

- **Message - текст сообщения**, которое появляется **в окне сообщения** при возникновении исключительной ситуации.
- **HelpContext** - содержит **контекст** справочной системы.

В файле проекта справочной системы можно задать контексты, указывающие на определенные тексты справки. Фактически, это ссылка на текст справки, предъявляемый в конкретной ситуации - контекстно.

Генерация исключительной ситуации

- При необходимости **можно создавать исключение любым из конструкторов**, или создать свой конструктор.
- Для стандартных ситуаций в модуле **SysUtils** и ряде других имеется достаточное число уже разработанных исключений, входящих в семейство класса **Exception**.
- Приведенные восемь конструкторов класса отличаются **способом формирования сообщения**.

Создание исключения

Объект-исключение можно создать с помощью одного из представленных **конструкторов**.

Создание соответствующего объекта и вызов нужного конструктора выполняется с помощью зарезервированного слова **raise** (вызывать):

raise *вызов конструктора*;

Пример

Необходимо получать сообщение об исключительной ситуации, суть которой в том, что некая переменная меньше 1:

```
.....
if N < 1
then raise Exception.Create
                    (‘Значение N меньше 1’);
{вызов конструктора}
.....
```

Примеры стандартных классов исключительных ситуаций

В библиотеках Delphi имеется достаточно классов, ответственных за обработку различных исключительных ситуаций.

- **Замечание**. В отличие от прочих типов, названия которых принято начинать с буквы **T**, имена этих классов начинаются с буквы **E** (Exception).

Имя класса	Когда возникает
EAbstractError	Попытка выполнения абстрактного метода
EAccessViolation	Обращение к недоступной области памяти (например, при выходе индекса за границы массива)
EConvertError	Попытка неверного преобразования типов (например, StrToInt(‘asd’)
EZeroDivide	Деление на ноль для вещественных чисел

EExternal Некорректное функционирование системы Windows

EInOutError Ошибка файлового ввода/вывода

EIntError Базовый класс для классов исключительных ситуаций при работе с целыми числами

EIntOverflow Недопустимо
большое значение
при работе
с целыми числами

EOverflow Переполнение
при работе с числами
с плавающей запятой

EOutOfMemory Нехватка памяти

Контроль над исключительными ситуациями

Язык Object Pascal предоставляет следующую конструкцию для контроля возникновения исключительных ситуаций:

try

Операторы;

except

Обрабатываемые классы исключительных ситуаций

else оператор

end;

Порядок выполнения операторов:

- Последовательно выполняются операторы, расположенные между **try** и **except**.
- В случае возникновения при выполнении какого-либо из них исключительной ситуации происходит обращение к списку классов, перечисленных между **except** и **else.**
  - Выполняется действие, указанное для соответствующего класса.
  - Если возникшая ситуация не относится ни к одному из явно обрабатываемых классов, то управление передается оператору секции **else.**
- Управление передается оператору, стоящему сразу после **end.**

Примечание. Секция **else** необязательна (сокращенная форма):

try

Операторы;

except

Обрабатываемые классы исключительных ситуаций

end;

Конструкция **try except end** -

структурированный оператор.

Как и любой оператор, **try except end** можно поместить внутри другого структурированного оператора, в том числе и аналогичного:

```
try
……
try
except
end
except
end
```

Запись классов, предназначенных для обработки:

on имя класса **do** операторы;

- Таких классов может быть **несколько**.
- Поиск класса подходящей исключительной ситуации осуществляется в **последовательном** порядке.
- Если возникшую ошибку можно отнести к нескольким классам, то будет вызван обработчик для класса, расположенного **первым** в секции except.

- Пример. Выполняется сложение целых чисел Y, Z:

```
try
X := Y + Z;
except
on EIntError do P1;
on EIntOverflow do P2;
end;
```

- Если при выполнении оператора
  X := Y + Z
  возникнет ошибка переполнения EIntOverflow,
  то, тем не менее, будет вызвана **P1**.
- Эта ошибка относится также и к классу EIntError, который указан **первым** в секции except.
- Т.е. нужно:

```
except
on EIntOverflow do P2;
on EIntError do P1;
end;
```

- В ряде случаев удобно использовать **единый** обработчик для любой исключительной ситуации.
- Блок try примет вид:

```
try
Операторы;
except
Операторы;
end;
```

- Например:

```
try
X := Y + Z;
except
MyErrorProc;
end;
```

- В случае возникновения любой исключительной ситуации при выполнении оператора X := Y + Z вызывается процедура MyErrorProc.

- **Если реализованной программистом обработки ошибки недостаточно, можно передать управление соответствующему стандартному обработчику Delphi с помощью зарезервированного слова raise.**

- Пример. Выполняется деление:

```
try
X := 100 div Y ;
except
on EDivByZero do
begin
ShowMessage (‘Ошибка’);
raise;
end;
end;
```

- После выдачи сообщения “Ошибка” исполнится стандартный обработчик ошибки EDivByZero.

- **Из секции, в которой располагаются обработчики исключительных ситуаций, можно генерировать другие исключительные ситуации.**
- **При этом слово raise используется для вызова соответствующего конструктора:**

raise имя класса.имя конструктора(парам.);

- Например:

```
try
X := Y + Z;
except
on EIntError do
raise EIntOverflow.Create (‘Возможно
    переполнение’);
end;
```

- Если при сложении Y, Z возникнет какая-то арифметическая ошибка (базовый класс EIntError), то в обработчике этой ошибки будет сгенерирована другая исключительная ситуация - EIntOverflow, а обработка EIntError завершится.

Выполнение завершающих действий

- В некоторых случаях программисту не нужен собственный обработчик ошибок
- **НО** требуется, чтобы даже в случае исключительной ситуации **программа гарантированно выполняла** те или иные действия.

- *Для программирования подобных алгоритмов используется конструкция вида:*

try
Операторы;
finally
Заключительные операторы;
end;

- *Заключительные действия будут выполнены в* **любом** *случае, независимо от возникновения исключительной ситуации при выполнении операторов секции try.*

Пример. Освобождение памяти, выделенной для динамического массива, происходит независимо от успешности его обработки:

```
var DynArr: array of integer;
…
try
SetLength(DynArr, 100000);
…..
{обработка массива}
….
finally
DynArr := NIL
end;
```

Передача объектов, связанных с исключительными ситуациями

- При возникновении исключительной ситуации специальный обработчик создает соответствующий ей объект.
- Для получения доступа к этому объекту требуется видоизменить конструкцию секции except:

on E: имя класса **do** операторы;

- *E – это произвольный идентификатор переменной (описывать ее не нужно), в которую запишется соответствующий объект.*
- *К ней можно обращаться в этой же секции, например, для изменения ее свойств.*

Например:

```
on E: EAccessViolation do
begin
E.Message := ‘Одна из самых непонятных
    ошибок’;
raise;
end;
```

- *С помощью свойства Message **определен нестандартный текст** сообщения о соответствующей исключительной ситуации.*