اجرای دستور به دستور : کنترل خطاها :

در بخش قبل خطاهایی را بررسی کردیم که کلا اتفاق می افتند چه خطاهایی که نقض الگوریتم است مثل عدم در نظر گرفتن احتمال تقسیم بر صفر دو جایی که این احتمال وجود دارد و چه خطاهایی که در اثر استفاده اشتباه از نرم افزار اتفاق می افتد. در واقع بخشی گذشته یک روش کلی برای پیدا کردن خطا های در برنامه و دسته بندی آنها بود. با روشهای فصل گذشته ریش های مشکلات برنامه پیدا می شد.

خطاهایی که نقض الگوریتم است با اصلاح آن از بین می رود و بر استحکام الگوریتم افزوده می شود اما باید برای خطاهای نوع کاربری هم فکر مقتضی کرد. این کار با " مدیریت استثناها " که قبلاً هم به آن اشاره کردیم ممکن است و به این معنی است که خطاهایی که استثنائاً در اجرای برنامه ایجاد می شوند را در هنگام رخداد به نحوی مناسب به اطلاع کاربر برسانیم و هر گونه مشکل احتمال در اثر بروز خطا را رفع کنیم از قبیل "نشت حافظه[1]" که عبارتست از اختصاص حافظه و عدم آزاد سازی آن که باعث می شود رشته مدیریت حافظه از دست سیستم عامل در رفته و دچار کمبودر حافظه گردد.

اولین و مهمترین ساختار در این مورد ساختار *try* است.

```
try
 //Some operation
except
 //Operations in of
 //Exceptions
end;
```

```
try
//Some operation
finally
//Must be done operation
end;
```

در مواردی می دانیم در اثر عدم استفاده صحیح کاربر احتمال خطا در بخش خاصی از برنامه وجود دارد مثلاً برنامه ای که باید اعداد را تغییر مبنا دهد؛

اگر مبنای وارد شده از طرف کاربر **0** باشد حتماً خطا ایجاد می شود. در چنین مواردی کدی را که احتمال ایجاد خطا دارد در بخش بین *try*,*except* یا *try*,*finally* قرار می دهیم. در نوع اول اگر خطا اتفاق بیافتد کنترل برنامه به بخش بین *except*,*end* منتقل می شود. و در ساختار دوم هر اتفاقی در اجرای کدهای بین *try*,*finally* چه خطا داشته باشیم و چه خیر بخش بین *finally* و *end* اجرا می شود.

در مورد اول که نیاز به مثال کاربرد نیست اما در مورد حالت دوم مثال زیر کاربرد آن را نشان می دهد:

فرض کنید قرار است که عددی از کاربر دریافت شود عدد قبلی بر آن تقسیم شود و در هر صورت (چه خطا رخ داد و چه خیر پیغام خاصی رو صفحه نمایش داده شود).

```
try
  X :X / i;
  M
finally
  ShowMessage;
end;
```

و یا در مواردی تخصصی تر هنگامی که یک شیئی ایجاد می شود حتماً باید فضای اختصاص داده شده به آن آزاد شود، *MemoryLeak age* ایجاد نشد یعنی چه خطا رخ داد و چه رخ نداد باید حافظه اختصاص داده شده آزاد شود.

```
X :=TObject.Create;
try
//Code that mayGererate errorss
finally
X .Free;
end;
```

با این ابزارهای بسیاری از خطاهای برنامه مدیریت می شوند البته نکات دیگری در استفاده از

try...except...end وجود دارد که اگر علاقه به دانستن آنها دارید؛ به راهنمای *Delphi* مراجعه کنید.

این ابزارها بسیاری از خطاها را پوشش می دهد برای باقیمانده خطاها باید ابتدا مفاهیمی را بدانیم تا مدیریتشان کنیم.

در دلفی اخبار رخدادن یک استثنا توسط شیئی هایی که از *Exception* به ارث برده شده اند اتفاق می افتد. این کلاسهای تقریباً هیچ چیزی علاوه بر *Exception* ندارند بلکه نام آنها متفاوت است لذا خاصیتهای *Exception* مشترک بین تمامی آنهاست. این اشیا خاصیت *Massage* دارند. خاصیت *Massage* از نوع رشته حرفی و مقدار پیغام خطی به زبان انگلیسی است.

برای مدیریت کلی خطاها ابتدا از *Component Pallette* بخش *Additional* جزء *Application Events* را انتخاب کرده و سپس رخداد *OnException* را بیابید و انتخاب کنید در متن زیر برنامه کارهای لازم برای مدیریت کلیه استثناها را انجام دهید. از جمله پارامترهای منتقل شده به برنامه یک نمونه از شیئی است که از محتویات آن باید خطا را تشخیص دهید.

البته در اینجا یکی از امکانات *Delphi* که نیازی به توضیح کلی آن دیده نمی شود مطرح می کنیم و آن را تطبیق نوع داده یا *typeCasting* است. در زبان دلفی اشیا می توانند به شیئی های والد خود تطبیق شوند مثلاًفرض کنید شیئی *EDivision By Zero* از کلاس *Exception* ارث برده شده،

```
var
  X := Exception;
  Y := EDivisionByZero;
begin
  X :=Y ;
end;
```

در مثال بالا محتویات یک متغیر از نوع *Exception* یک نمونه از شیئی *EDivision By Zero* است که به این عمل *typeCasting* گویند. اما برای آنکه متوجه شدید که آیا یک نمونه متغیز از نوع *Exception* از روی *EDivisionByZero* تطبیق شده یا خیر از عبارت زیراستفاده کنید *X is EDivisionByZero* مقدار منطقی بر می گرداند که اگر تطبیق شده باشد مقدار "درست" و در غیر اینصورت "*false*" بر می گرداند. به این ترتیب می توانید خطاهای خاص مثلاً خطای دسترسی غیر مجاز به حافظه که *EAccessViolation* است را تشخیص دهید.

```
if E is EAccessViolation then
begin
M
end;
```

با بحث قبل مبحث مدیریت استثنائات تقریباً به پایان می رسد.

فقط یک نکته باقیمانده آن باعث شدن یک استثنائات به این معنی که به مجموعه برنامه بفهمانیم که یک استثنای خاص رخ داده.

در وحله اول مساله تعریف انواع استثنای جدید است. این کار در بخش صورت می گیرد و در حد کارهای ما تعریف واقعاً و بدون هیچ دستکاری فرمت زیر است

```
type
< ExceptionName >= Class(Exception)
public:
Code: Integer;
end;
```

که در قرار دادن کد بصورت فوق فقط صوری است.

حال هر کجا که خطای مورد نظر پیش آمد با دستور زیر استثنا را به اطلاع برنامه می رسانیم.

raise < ExceptionName >;

این کار معادل اینست که در هر بار بر خورد با خطای فوق کار خاصی انجام دهیم ولی با روش بالا فقط یکبار بررسی و اعمال مناسب رفع و اطلاع خطاها را انجام داده و آن کار را در مکانی که کلیه خطاها را مدیریت می کنیم انجام می دهیم و این برای خوانا شدن برنامه مفید است. لذا فراخوانی *raise* معادل است با باشد. فراخوانی زیر برنامه مربوط به *OnException* با پارامتر *E* که مقدار آن *< ExceptionName >* باشد.

Memory leakage [1]