Chapter

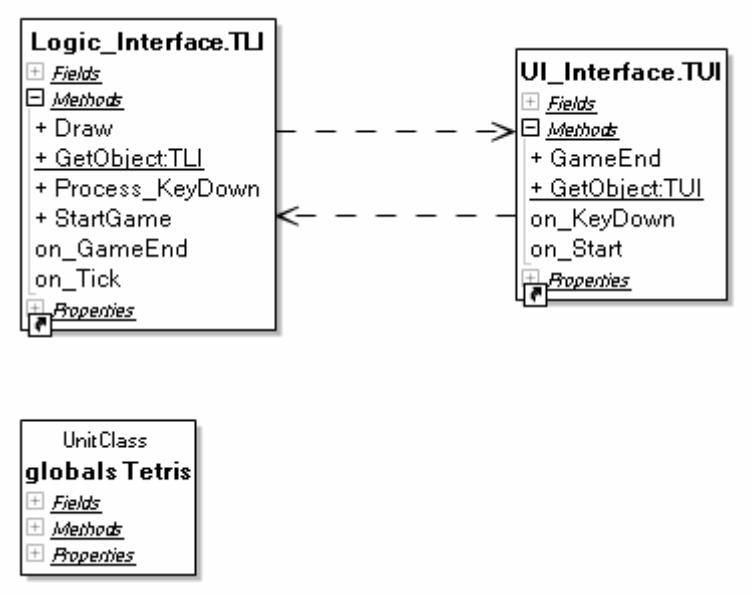
0

테트리스 게임 만들기

1단계 - UI와 Logic의 분리

이번 강의에서는 필자가 원래 사용하던 문서작성법이 아닌 클래스 다이어그램만으로 기능설계와 구조설계를 병행하도록 하겠다.

User-Interface의 기능분석



[그림 1] UI와 Logic의 분리

[그림 1]에서 우선 우측의 UI_Interface.TUI 클래스를 주목하자. 해당 클래스의 멤버들에서 사용된 작명규칙을 보면, Public 메쏘드 앞에는 아무런 접두어가 붙지 않지만, 이벤트에 해당하는 메소드 앞에는 “on_”을 붙여놨다. 이것은 필자만의 습관으로, 이름만으로 해당 메쏘드의 역할을 쉽게 구별하기 위해서이다. 추후 내부처리용으로 사용되는 Private 메쏘드는 “do_”를 앞에 붙이게 된다.

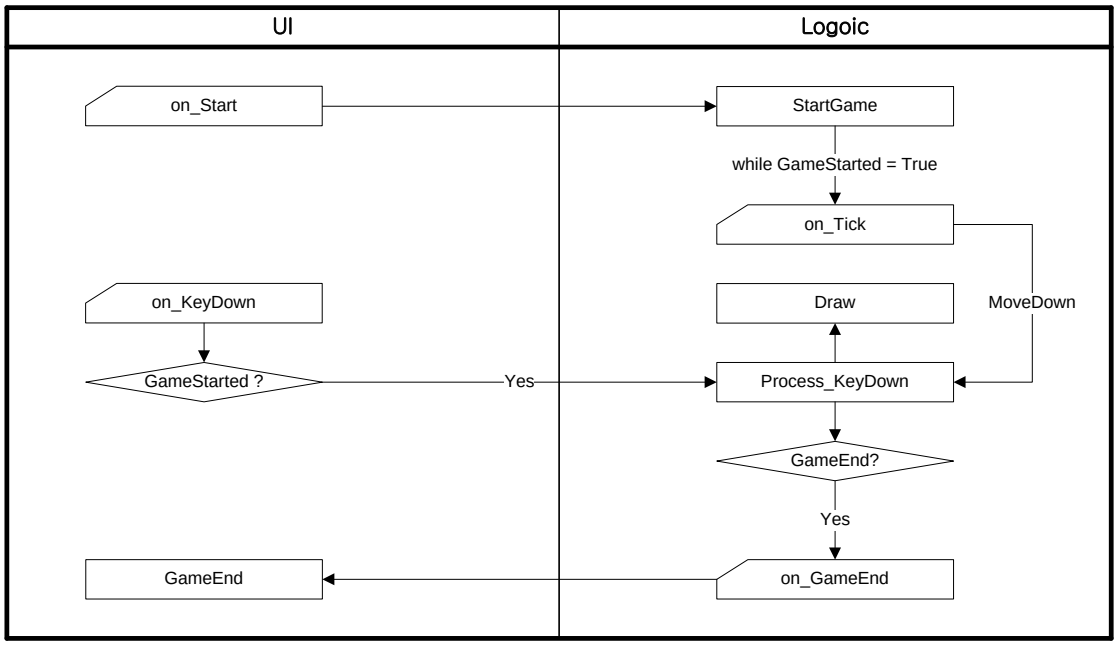
[그림 1]의 UI_Interface.TUI 클래스는 아래와 같은 기능을 가져야 한다.

- 기능
  - GameEnd : 게임 종료 처리
- 이벤트
  - on_Start : 사용자가 게임을 시작하도록 하였음
  - on_KeyDown : 사용자가 키보드를 클릭하였음

Logic-Interface의 기능분석

이어서 [그림 1]의 좌측 Logic_Interface.TLI 클래스는 테트리스 게임의 논리계층의 최상의 클래스이며, 해당 클래스는 다음과 같은 기능을 가져야 한다.

- 기능
  - StartGame
  - Process_KeyDown
  - Draw
- 이벤트
  - on_Tick
  - on_GameEnd



[그림 2] Job Flow

[그림 2]에서는 UI와 Logic이 서로 호출하는 순서와 이벤트의 흐름을 설명하고 있다.

on_Start이벤트가 발생하면 Logic의 GameStarted 속성을 True로 변경한다. GameStarted 속성이 True로 변경되면 내부 타이머가 작동하면서 주기적으로 on_Tick 이벤트를 발생한다.

On_Tick 이벤트는 블록을 한 칸씩 밑으로 이동하는 메소드를 호출한다. 이동을 담당하고 있는 Process_KeyDown 메소드는 더 이상 블록을 내려놓을 수 없을 때, GameStarted 속성을 False로 변경시키면서 on_GameEnd 이벤트를 발생시킨다.

사용자가 키를 누르게 되면 on_KeyDown 이벤트가 발생하게 되고, 게임 중일 경우에만 Logic에게 이 메시지를 전달하게 된다.

사실 기능분석이 분석의 가장 기본이 되기는 하지만, 업무분석에는 동적분석이 더욱 효율적일 때가 많다. 어느 분석을 먼저 하더라도 분석은 단일 프로세스로 끝나는 경우는 거의 없다. 즉, 기능분석을 마치고 동적분석을 하는 동안 필요한 기능이 분석되지 않은 것을 발견했을 경우 다시 기능분석을 변경하고, 동적분석 후에도 기능분석을 통해서 동적분석의 오류를 발견할 수도 있다.

기능분석과 구조분석 그리고 동적분석은 진행하는 동안 서로 보완될 경우가 많다. 그리고, 반드시 “어느 것을 먼저 해야 한다”라는 규칙은 없다.

UI_Interface Unit의 소스

```
 1 : unit UI_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils;
 7 :
 8 : type
 9 :   TUI = class
10 :   protected
11 :     procedure on_KeyDown(Key:Word);
12 :     procedure on_Start;
13 :   public
14 :     class function GetObject:TUI;
15 :     procedure GameEnd;
16 :   end;
17 :
18 : implementation
19 :
20 : uses
21 :   Logic_Interface;
22 :
23 : var
24 :   MyObject : TUI = Nil;
25 :
26 : { TUI }
27 :
28 : class function TUI.GetObject: TUI;
29 : begin
30 :   if MyObject = Nil then MyObject:= TUI.Create;
31 :   Result:= MyObject;
32 : end;
33 :
```

```
34 : procedure TUI.on_Start;
35 : begin
36 :   TLI.GetObject.StartGame;
37 : end;
38 :
39 : procedure TUI.on_KeyDown(Key:Word);
40 : begin
41 :   if TLI.GetObject.GameStarted = True then TLI.GetObject.Process_KeyDown(Key);
42 : end;
43 :
44 : procedure TUI.GameEnd;
45 : begin
46 :   {ToDo : }
47 : end;
48 :
49 : end.
```

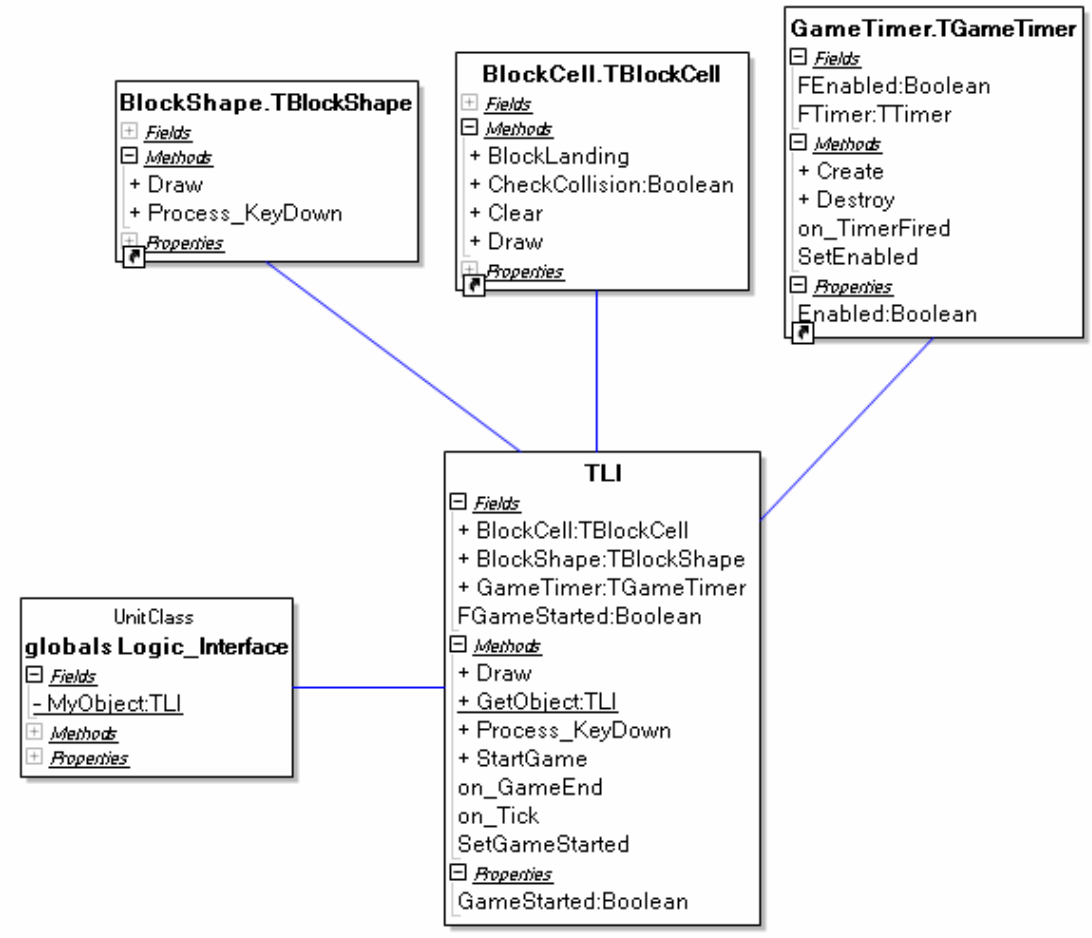
Logic_Interface Unit의 소스

```
 1 : unit Logic_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   BlockShape, BlockCell, GameTimer, Windows, Classes, SysUtils;
 7 :
 8 : type
 9 :   TLI = class
10 :   private
11 :     FGameStarted: Boolean;
12 :     procedure SetGameStarted(const Value: Boolean);
13 :   protected
14 :     procedure on_GameEnd;
15 :     procedure on_Tick;
16 :   public
17 :     GameTimer : TGameTimer;
18 :     BlockCell : TBlockCell;
19 :     BlockShape : TBlockShape;
20 :     class function GetObject:TLI;
21 :     procedure StartGame;
22 :     procedure Draw;
23 :     procedure Process_KeyDown(Key:Word);
24 :   published
25 :     property GameStarted : Boolean read FGameStarted write SetGameStarted;
26 :   end;
27 :
28 : implementation
29 :
30 : uses
31 :   UI_Interface;
32 :
33 : var
```

```
34 :   MyObject : TLI = Nil;
35 :
36 : { TLI }
37 :
38 : class function TLI.GetObject: TLI;
39 : begin
40 :   if MyObject = Nil then MyObject:= TLI.Create;
41 :   Result:= MyObject;
42 : end;
43 :
44 : procedure TLI.SetGameStarted(const Value: Boolean);
45 : begin
46 :   FGameStarted := Value;
47 :
48 :   if Value = True then Self.StartGame
49 :   else Self.on_GameEnd;
50 : end;
51 :
52 : procedure TLI.StartGame;
53 : begin
54 :   {ToDo : }
55 : end;
56 :
57 : procedure TLI.on_GameEnd;
58 : begin
59 :   TUI.GetObject.GameEnd;
60 : end;
61 :
62 : procedure TLI.on_Tick;
63 : begin
64 :   Self.Process_KeyDown(VK_Down);
65 : end;
66 :
67 : procedure TLI.Draw;
68 : begin
69 :   {ToDo : }
```

```
70 : end;
71 :
72 : procedure TLI.Process_KeyDown(Key:Word);
73 : begin
74 :   case Key of
75 :       VK_Left :  {ToDo : };
76 :       VK_Right : {ToDo : };
77 :       VK_Up :    {ToDo : };
78 :       VK_Down :  {ToDo : if 게임종료 then GameStarted:= False};
79 :       VK_Space : {ToDo : 바닥까지 VK_Down 처리};
80 :   end;
81 :
82 :   Self.Draw;
83 : end;
84 :
85 : end.
```

기능분석



[그림 3] Logic Interface Class Diagram

TGameTimer의 기능목록

- 기능
  - 없음
- 이벤트
  - on_TimerFired

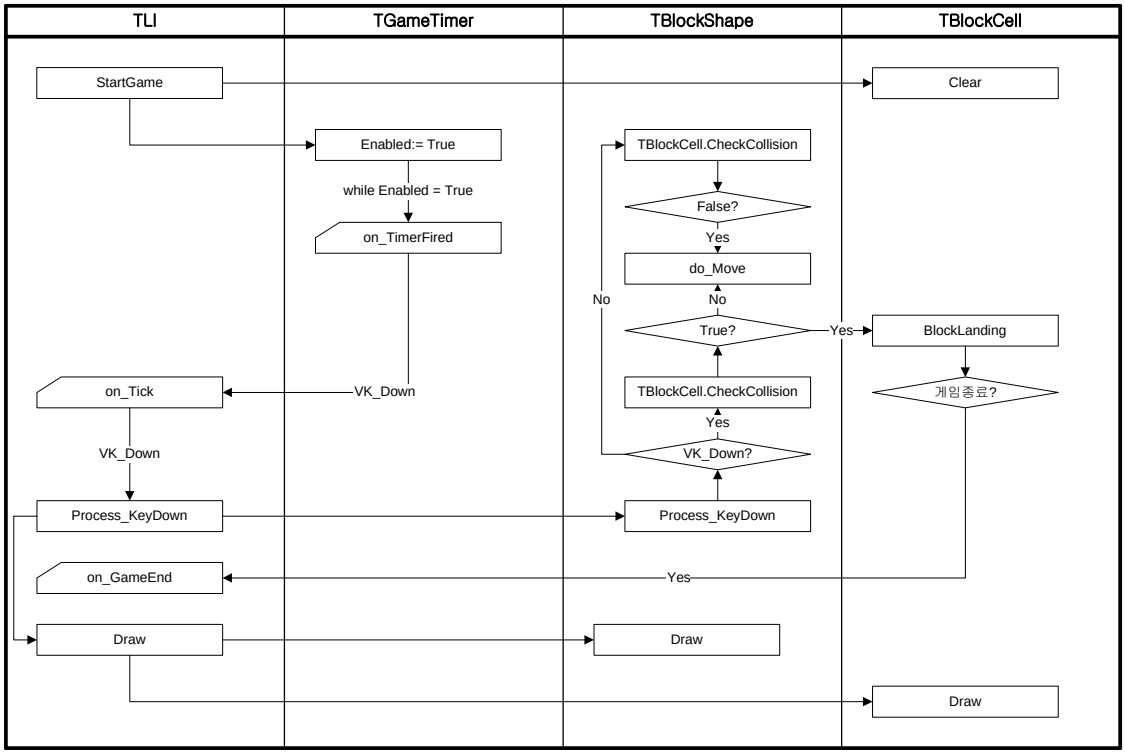
TBlockShape의 기능목록

- 기능
  - Draw
  - Process_KeyDown
- 이벤트
  - 없음

TBlockCell의 기능목록

- 기능
  - Clear
  - Draw
  - BlockLanding
  - Checkcollision
- 이벤트
  - 없음

동적분석



[그림 4] Job Flow

[그림 4]에서는 동적분석을 통해서 Logic 부분의 전반적인 흐름을 표현하고 있다. 여기서는 기능 분석에서 발견하지 못했던 "do_Move" 메소드가 필요하다는 것을 발견하였다. 이제 기능분석을 통해 작성한 클래스 다이어그램을 수정하여야 한다.

이것은 소스상에서만 반영하도록 하겠다. 현재 필자가 사용하고 있는 BDS 2006에서는 소스를 변경하면 클래스 다이어그램이 자동으로 변경되기 때문에 변경작업에 의한 문서작성에 대해서는 스트레스 받을 필요가 없다.

또한, "do_Move"는 메쏘드 이름에서 알 수 있듯이 Private 메쏘드이다. 따라서, 기능분석 동안에 찾아내기는 쉽지 않다.

Logic_Interface Unit의 수정

```
 1 : unit Logic_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   BlockShape, BlockCell, GameTimer, Windows, Classes, SysUtils;
 7 :
 8 : type
 9 :   TLI = class
10 :   private
11 :     FGameStarted: Boolean;
12 :     procedure SetGameStarted(const Value: Boolean);
13 :   protected
14 :     procedure on_GameEnd;
15 :     procedure on_Tick;
16 :   public
17 :     GameTimer : TGameTimer;
18 :     BlockCell : TBlockCell;
19 :     BlockShape : TBlockShape;
20 :     class function GetObject:TLI;
21 :     procedure StartGame;
22 :     procedure Draw;
23 :     procedure Process_KeyDown(Key:Word);
24 :   published
25 :     property GameStarted : Boolean read FGameStarted write SetGameStarted;
26 :   end;
27 :
28 : implementation
29 :
30 : uses
31 :   UI_Interface;
32 :
33 : var
```

```
34 :   MyObject : TLI = Nil;
35 :
36 : { TLI }
37 :
38 : class function TLI.GetObject: TLI;
39 : begin
40 :   if MyObject = Nil then MyObject:= TLI.Create;
41 :   Result:= MyObject;
42 : end;
43 :
44 : procedure TLI.SetGameStarted(const Value: Boolean);
45 : begin
46 :   FGameStarted := Value;
47 :
48 :   if Value = True then Self.StartGame
49 :   else Self.on_GameEnd;
50 : end;
51 :
52 : procedure TLI.StartGame;
53 : begin
54 :   GameTimer.Enabled:= True;
55 : end;
56 :
57 : procedure TLI.on_GameEnd;
58 : begin
59 :   GameTimer.Enabled:= False;
60 :   TUI.GetObject.GameEnd;
61 : end;
62 :
63 : procedure TLI.on_Tick;
64 : begin
65 :   Self.Process_KeyDown(VK_Down);
66 : end;
67 :
68 : procedure TLI.Draw;
69 : begin
```

```
70 :   BlockCell.Draw;
71 :   BlockShape.Draw;
72 : end;
73 :
74 : procedure TLI.Process_KeyDown(Key:Word);
75 : begin
76 :   BlockShape.Process_KeyDown(Key);
77 :   Self.Draw;
78 : end;
79 :
80 : end.
```

GameTimer Unit의 소스

```
 1 : unit GameTimer;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, ExtCtrls;
 7 :
 8 : type
 9 :   TGameTimer = class
10 :   private
11 :     FTimer : TTimer;
12 :     function GetEnabled: Boolean;
13 :     procedure SetEnabled(const Value: Boolean);
14 :   protected
15 :     procedure on_TimerFired(Sender:TObject);
16 :   public
17 :     constructor Create;
18 :     Destructor Destroy; override;
19 :   published
20 :     property Enabled : Boolean read GetEnabled write SetEnabled;
21 :   end;
22 :
23 : implementation
24 :
25 : uses
26 :   Logic_Interface;
27 :
28 : type
29 :   TFLI = class(TLI)
30 :   end;
31 :
32 : { TGameTimer }
33 :
```

```
34 : constructor TGameTimer.Create;
35 : begin
36 :   inherited;
37 :
38 :   FTimer:= TTimer.Create(Nil);
39 :   FTimer.Interval:= 50;
40 :   FTimer.OnTimer:= Self.on_TimerFired;
41 :   FTimer.Enabled:= False;
42 : end;
43 :
44 : destructor TGameTimer.Destroy;
45 : begin
46 :   FTimer.Free;
47 :
48 :   inherited;
49 : end;
50 :
51 : procedure TGameTimer.on_TimerFired(Sender: TObject);
52 : begin
53 :   TFLI(TFLI.GetObject).on_Tick;
54 : end;
55 :
56 : function TGameTimer.GetEnabled: Boolean;
57 : begin
58 :   Result:= FTimer.Enabled;
59 : end;
60 :
61 : procedure TGameTimer.SetEnabled(const Value: Boolean);
62 : begin
63 :   FTimer.Enabled:= Value;
64 : end;
65 :
66 : end.
```

TBlockShape Unit의 소스

```
 1 : unit BlockShape;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, Windows;
 7 :
 8 : type
 9 :   TBlockShape = class
10 :   public
11 :     procedure Draw;
12 :     procedure Process_KeyDown(Key:Word);
13 :   private
14 :     function do_MoveDown:Boolean;
15 :     procedure do_Drop;
16 :     procedure do_Move(Key:Word);
17 :   end;
18 :
19 : implementation
20 :
21 : uses
22 :   Logic_Interface;
23 :
24 : procedure TBlockShape.Draw;
25 : begin
26 :   {Todo : }
27 : end;
28 :
29 : procedure TBlockShape.Process_KeyDown(Key:Word);
30 : begin
31 :   if Key = VK_Down then do_MoveDown
32 :   else do_Move(Key);
33 : end;
```

```
34 :
35 : function TBlockShape.do_MoveDown:Boolean;
36 : begin
37 :   Result:= not TLI.GetObject.BlockCell.CheckCollision(VK_Down);
38 :
39 :   if Result = True then
40 :     {Todo : }
41 :   else
42 :     TLI.GetObject.BlockCell.BlockLanding;
43 : end;
44 :
45 : procedure TBlockShape.do_Drop;
46 : var
47 :   bMoved : Boolean;
48 : begin
49 :   Repeat
50 :     bMoved:= do_MoveDown;
51 :   until bMoved = False;
52 : end;
53 :
54 : procedure TBlockShape.do_Move(Key:Word);
55 : begin
56 :   if TLI.GetObject.BlockCell.CheckCollision(Key) = True then Exit;
57 :
58 :   case Key of
59 :       VK_Left :  {ToDo : };
60 :       VK_Right : {ToDo : };
61 :       VK_Up :    {ToDo : };
62 :       VK_Space : do_Drop;
63 :   end;
64 : end;
65 :
66 : end.
```

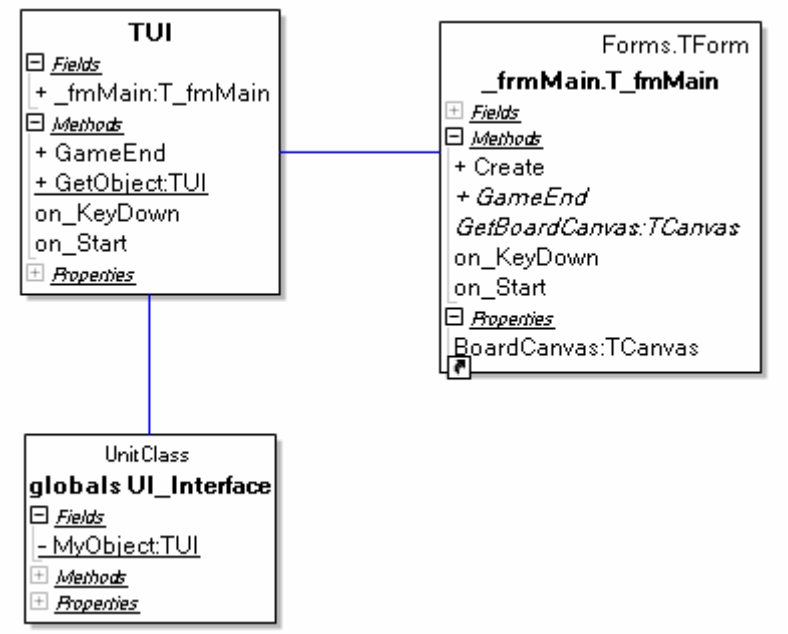
[그림 4]의 Job Flow와는 달리 좀더 로직을 쉽게 접근할 수 있도록 Private 메쏘드 몇 개가 추가되었다. Flow 자체는 동일하다.

TBlockCell Unit의 소스

```
 1 : unit BlockCell;
 2 :
 3 : interface
 4 :
 5 : type
 6 :   TBlockCell = class
 7 :   public
 8 :     procedure Draw;
 9 :     procedure Clear;
10 :     procedure BlockLanding;
11 :     function CheckCollision(Key:Word):Boolean;
12 :   end;
13 :
14 : implementation
15 :
16 : uses
17 :   Logic_Interface;
18 :
19 : procedure TBlockCell.Draw;
20 : begin
21 :   {Todo : }
22 : end;
23 :
24 : procedure TBlockCell.Clear;
25 : begin
26 :   {Todo : }
27 : end;
28 :
29 : procedure TBlockCell.BlockLanding;
30 : begin
31 :   {Todo : }
32 :
33 :   {Todo : 종료조건 처리}
```

```
34 :   if True then TLI.GetObject.GameStarted:= False;
35 : end;
36 :
37 : function TBlockCell.CheckCollision(Key:Word):Boolean;
38 : begin
39 :   {Todo : }
40 : end;
41 :
42 : end.
```

기능분석



[그림 5] User Interface 클래스 다이어그램

TUI 클래스는 변경된 사항이 없으니, _fmMain 클래스의 기능목록만 살펴보자.

- 기능
  - GameEnd
- 이벤트
  - on_KeyDown
  - on_Start

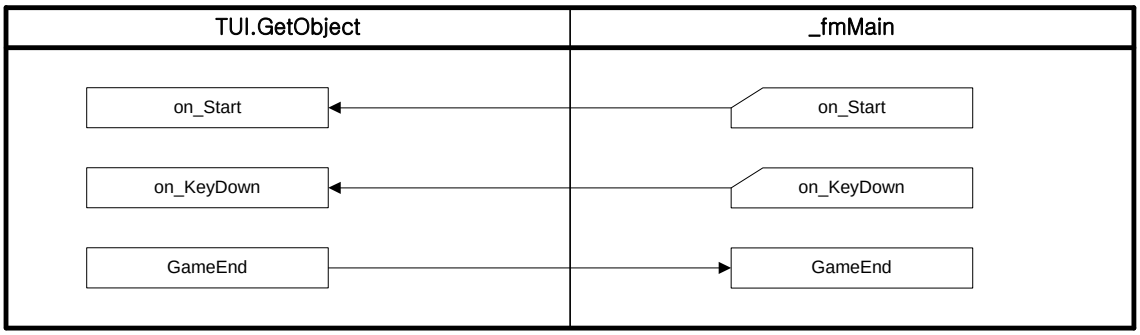
_fmMain은 메인폼에 대한 부모클래스이다. TUI는 직접 메인폼에 대하여 의존하지 않도록 중간에 _fmMain 클래스를 생성하고, 메인폼은 이것을 상속받아서 사용하도록 한다. 상속받아서 처리하는 이유는 TUI 자체의 논리계층과 구현계층을 분리하여 변경사항이 발생하더라도 변경사항에 대한 쇼크를 줄이기 위해서이다.

예를 들어 TLI에서 게임이 종료되었음을 알리면 TUI.GameEnd가 호출된다. 이때 TUI는 _fmMain.GameEnd라는 **abstract** 메쏘드를 실행하게 된다. 추후 메인폼이 _fmMain에서 상속을

받아서 해당 메쏘드를 override 하여 게임 종료처리를 완료하게 되는 것이다.

메인폼이 게임 종료처리를 어떠한 방식으로 진행하던지 이제 TUI는 알 필요가 없는 것이다. 그것은 TUI가 메인폼이 아닌 _fmMain에만 의존관계를 가지고 있기 때문이다.

동적분석



[그림 6] Job Flow

현재의 설계는 _fmMain에서 on_Start 등의 이벤트가 발생하면 TUI의 같은 on_Start가 호출되어 중복된 메쏘드가 나오게 된다. 이것은 Demeter 법칙을 준수하기 위한 것이다. TLI가 TUI이 너무 깊숙한 곳까지 알게 되면 소스코드의 결합도가 높아지기 때문이다. 하지만, 필자는 TUI가 너무 많은 메쏘드를 처리하게 되는 시점에서 모든 메쏘드는 TUI 하위 클래스에게 위임한다. 하나의 클래스에 너무 많은 메쏘드를 포함시키면 가독성이 떨어지기 때문이다. 일반적으로는 다른 클래스의 내장을 후벼 파는 것은 소스의 유연성을 떨어트리게 된다.

객체지향적 설계에서는 때에 따라서 그 적용방법을 달리해야 할 때가 발생한다. "반드시"라는 법칙은 없다. (어쩌면 그것이 다행일지도 모르겠다. 만약 "반드시"라는 법칙이 통한다면 사람이 아닌 기계가 대신 설계를 도맡아 할지도 모르겠다.)

UI_Interface Unit의 수정

```
 1 : unit UI_Interface;
 2 :
 3 : interface
 4 :
 5 : uses
 6 :   Classes, SysUtils, _frmMain;
 7 :
 8 : type
 9 :   TUI = class
10 :   protected
11 :     procedure on_KeyDown(Key:Word);
```

```
12 :    procedure on_Start;
13 :  public
14 :    _fmMain : T_fmMain;
15 :    class function GetObject:TUI;
16 :    procedure GameEnd;
17 :  end;
18 :
19 : implementation
20 :
21 : uses
22 :  Logic_Interface;
23 :
24 : var
25 :  MyObject : TUI = Nil;
26 :
27 : { TUI }
28 :
29 : class function TUI.GetObject: TUI;
30 : begin
31 :  if MyObject = Nil then MyObject:= TUI.Create;
32 :  Result:= MyObject;
33 : end;
34 :
35 : procedure TUI.on_Start;
36 : begin
37 :  TLI.GetObject.StartGame;
38 : end;
39 :
40 : procedure TUI.on_KeyDown(Key:Word);
41 : begin
42 :  if TLI.GetObject.GameStarted = True then TLI.GetObject.Process_KeyDown(Key);
43 : end;
44 :
45 : procedure TUI.GameEnd;
46 : begin
47 :  Self._fmMain.GameEnd;
```

```
48 : end;
49 :
50 : end.
```

_fmMain Unit의 소스

```
 1 : unit _frmMain;
 2 :
 3 : interface
 4 : uses
 5 :   Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms;
 6 :
 7 : type
 8 :   T_fmMain = class(TForm)
 9 :   protected
10 :     function GetBoardCanvas: TCanvas; virtual; abstract;
11 :     procedure on_KeyDown(Key:Word);
12 :     procedure on_Start;
13 :   public
14 :     constructor Create(AOwner:TComponent); override;
15 :     procedure GameEnd; virtual; abstract;
16 :   published
17 :     property BoardCanvas : TCanvas read GetBoardCanvas;
18 :   end;
19 :
20 : implementation
21 :
22 : uses
23 :   UI_Interface;
24 :
25 : type
26 :   TFUI = class(TUI)
27 :   end;
28 :
29 : { T_fmMain }
30 :
31 : constructor T_fmMain.Create(AOwner: TComponent);
32 : begin
33 :   inherited;
```

```
34 :
35 :   TUI.GetObject._fmMain:= Self;
36 : end;
37 :
38 : procedure T_fmMain.on_KeyDown(Key: Word);
39 : begin
40 :   TFUI(TUI.GetObject).on_KeyDown(Key);
41 : end;
42 :
43 : procedure T_fmMain.on_Start;
44 : begin
45 :   TFUI(TUI.GetObject).on_Start;
46 : end;
47 :
48 : end.
```

중간점검

필자는 하나의 프로젝트(델파이에서는 *.dpr) 파일에 관련된 범위를 프로세스 모듈이라고 부른다. 그리고, 여기까지가 프로세스 모듈에 대한 논리계층의 설계과정이다. 이제부터는 구현계층에 대해서 설명하고자 한다.

만약 프로세스 모듈에 대한 논리계층이 변경되어야 한다면, 그것은 업그레이드가 아닌 버전업의 시점이라고 필자는 생각한다. 구현계층이 이루어져 있는 상황에서 논리계층을 변경하는 것은 건물을 거의 지어놓은 상태에서 철골구조를 새로 하려는 것과 같다. (실제 프로젝트에서는 그것보다 더 위험하다.)