**WinApi**

**API Handle**

- 운영체제가 무언가를 식별하기 위한 키 값을 핸들이라고 함.

- 핸들은 보통 32bit 정수형 값을 가지고 있음 => 정수형 비교가 가장 빠르기 때문.

- 핸들번호는 운영체제가 정함.

- 핸들은 절대 중복되는 정보를 가지지 않음.

- Windows는 윈도우 창마다 ID값을 부여함. => **HWND**(윈도우 핸들)

- HWND : 핸들 윈도우의 약자. 윈도우의 핸들번호를 저장해서 사용.

하나의 프로그램에서 여러 윈도우를 띄울 수 있으며,

각 윈도우창의 번호(HWND)로 구분하는 것.

- HINSTANCE : 핸들 인스턴스, 프로그램의 인스턴스 식별자. 프로그램 자체의 실체화된 주소임.

**\* HINSTANCE는 프로그램 자체의 핸들이며, HWND는 프로그램안의 윈도우창의 번호임.**

**HDC**

- HDC -> Device Context를 만들어서 ID를 반환함.

\* Device Context? – 화면에 그리기 작업(렌더링)을 수행하는데 필요한 데이터의 집합체.

- 연필, 브러쉬 등 그리기 작업을 수행할 때 필요한 집합체임. 윈도우에 무엇을 그릴때는 항상 DC를 가져와서 그려야 함.

- DC의 목적지는 HWND, DC의 펜은 기본 펜(Black), DC의 브러쉬는 기본 브러쉬(White)임.

\* 커널 오브젝트? - Windows 운영체제에서 리소스(Resource : 프로세스, 쓰레드, 파일)들을

관리하기 위한 데이터를 저장하는 메모리 블록

**Pixel**

- 우리가 화면으로 볼 수 있는 것은 모니터라는 출력장치를 통해 볼 수 있는 것.

- 우리가 보고 있는 화면은 한 칸 한 칸의 pixel로 이루어져 있고, 기본 단위는 pixel.

- 한 pixel의 값은 R, G, B 각각 1byte(0~255)씩 메모리에 위치함. 1pixel = 3byte.

**BeginPaint()**

**-** HDC를 생성할 수 있는 함수.

- EndPaint()와 짝.

- 메시지 처리방식에서만 사용할 수 있는 전용함수라고 생각해야 함.

- 게임루프에 사용하기에는 부적절.

**윈도우 이벤트**

- WM\_PAINT : 윈도우에 무효화 영역(invalidate)이 발생한 경우 실행.

\* 무효화 영역 : 윈도우 창을 “최소화” 하면 발생.

\* WM\_PAINT를 강제로 발생하는 함수가 있음.

\* InvalidateRect([윈도우 핸들], nullptr, [이전 그리기를 지울지 여부]);

- WM\_KEYDOWN

WPARAM – 키다운시, 키 입력된 키 정보를 아스키 코드 값으로(대소문자 구분)넘김.

LPARAM – 마우스 클릭 시, 클라이언트 기준으로 마우스 좌표를 반환.

[ x(2byte), y(2byte) ] , LOWORD() - X좌표, HIWORD() – Y좌표.

- WM\_LBUTTONDOWN

\* 이 외에도 다양한 이벤트가 존재함. => MSDN 참조.

**(실습)사각형 그리기**

|  |
| --- |
| case WM\_PAINT: //무효화 영역이 발생한 경우.  {  PAINTSTRUCT ps;  HDC hdc = BeginPaint(hWnd, &ps); // DeviceContext(그리기와 관련.)  //1. 펜, 브러쉬 생성  HPEN hRedPen = CreatePen(PS\_DASHDOT, 1, RGB(255, 0, 0));  HBRUSH hRedBrush = (HBRUSH)GetStockObject(LTGRAY\_BRUSH);  //2. 펜, 브러쉬 설정  // SelectObejct()  // 내가 만든 Pen 혹은 Brush를 HDC에 설정하고,  // 이전에 사용하던 Pen, Brush를 void형 포인터로 반환.  HPEN hDefaultPen = (HPEN)SelectObject(hdc, hRedPen);  HBRUSH hDefaultBrush = (HBRUSH)SelectObject(hdc, hRedBrush);    //3. 사각형 그리기  Rectangle(hdc, 10, 10, 110, 110);    //4. 다시 default로 되돌리기.  SelectObject(hdc, hDefaultPen);  SelectObject(hdc, hDefaultBrush);  //5. 만들었던 펜, 브러쉬 삭제.  DeleteObject(hRedPen);  DeleteObject(hRedBrush);      EndPaint(hWnd, &ps);  }  break; |

**(실습)사각형 이동하기**

**(실습)사각형 그리기2**

**게임의 프레임**

- 일반적으로 게임은 Tick을 사용해 초당 60프레임 이상의 업데이트를 수행하는 프로그램임.

- 사람은 60프레임 이상은 되어야 자연스럽다고 느낀다.

**GetMessage vs PeekMessage**

**-** GetMessage

메시지 큐에서 메시지 확인 될 때까지 대기상태.

Msg.message == WM\_QUIT인 경우 false를 반환. = > 프로그램 종료

- PeekMessage

메시지 유무와 관계 없이 반환.

메시지 큐에서 메시지를 확인한 경우 true, 아니라면 false반환.

\* 윈도우의 메시지 처리시간은 전체 프로그램 실행 기간 동안 매우 일부에 불과함.

[GetTickCount() 메소드를 통해 확인해볼 수 있음. => GetTickCount()는 1초에 1000번 호출됨.]

\* 따라서, 메시지를 처리하는 시간 외의 시간을 게임 로직에 사용하기 위해

메시지 처리를GetMessage가아닌, PeekMessage로 검사 처리하여 사용함.

**\_Inout\_, \_in\_**

- \_Inout\_ => 값을 받기도 하고, 접근하여 결과값을 되돌리기도 함.

- \_In => 값을 받기만 함.

**AdjustWindowRect([LPRECT], [윈도우 스타일(WS\_OVERAPPEDWINDOW], [메뉴창 크기 계산 여부])**

- AdjustWindowRect 함수는 첫번째 인자로 넣어준 LPRECT를 통해 RECT의 값을 수정함.

- 윈도우 창, 메뉴바 두께도 계산하여 자동으로 rt의 값을 세팅.

**해상도에 맞게 윈도우 크기 조절.**

|  |
| --- |
| int CCore::Init(HWND \_hWnd, POINT \_ptResolution)  {  m\_hWnd = \_hWnd;  m\_ptResolution = \_ptResolution;    //해상도에 맞게 윈도우 크기를 조절.  RECT rt = {0, 0, \_ptResolution.x, \_ptResolution.y};  AdjustWindowRect(&rt, WS\_OVERLAPPEDWINDOW, false);  SetWindowPos(m\_hWnd, nullptr, 100, 100, rt.right - rt.left, rt.bottom - rt.top, 0);  return S\_OK;  } |

**GetDC([윈도우 핸들])**

- BeginPaint()를 대체하여 DC를 얻는 함수.

- ReleaseDC([윈도우 핸들], [DC])와 짝.