

화면 출력

HCI Programming 2 (321190)

2008년 가을학기

10/7/2008

박경신

Overview

- 윈도우의 화면 출력 원리 이해
 - GDI (Graphic Device Interface)
 - DC (Device Context)
- CDC 클래스를 이용한 화면 출력 기법
 - 다양한 DC 클래스
 - GDI 그래픽 함수
- 각종 GDI 객체를 생성하고 사용하는 방법
 - GDI Object (CPen, CBrush, CFont, 등등)
 - Stock GDI Object

2

GDI와 디바이스 컨텍스트

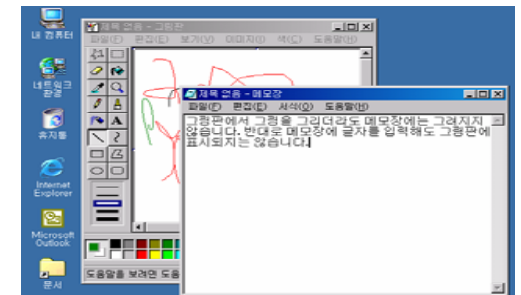
- 출력 시스템을 설계할 때 고려할 사항
 - 모니터, 비디오 카드, 프린터 등 출력에 사용되는 주변 장치가 변경되는 경우에도 프로그램을 수정할 필요가 없어야 함
 - 여러 프로그램이 화면을 분할해서 사용하므로 하나의 프로그램이 출력을 하는데 있어서 제약을 가해야 함 - 즉, 화면이나 기타 출력 장치를 직접 접근(Direct Access)하거나 독점해서 사용하는 것을 운영체제 차원에서 막아야 함

3

GDI와 디바이스 컨텍스트

- 화면 출력할 때 고려할 사항
 - 클라이언트 영역에 출력하려면 화면에 해당하는 윈도우 위치를 알아야 함
 - 화면에 여러 개의 윈도우가 있을 때 출력 결과가 다른 윈도우의 영역을 침범하지 않아야 함
 - 현재 출력할 화면이 다른 윈도우에 가려졌다면 출력을 할 수 없어야 함

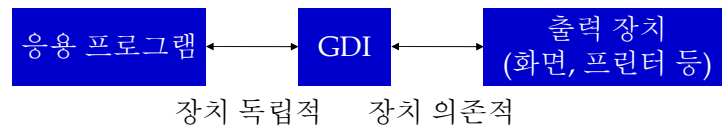
Device Context (DC) 사용



GDI와 디바이스 컨텍스트

□ GDI (Graphics Device Interface)

- 운영체제의 하위 시스템 중 하나로 DLL로 존재 (GDI32.DLL)
- 응용 프로그램의 요청을 받아서 **실제 출력 장치에 대한 출력 담당**
- 응용 프로그램에 대한 독립적인 그래픽 동작 수행
- 장치에 의존하지 않으므로 장치가 변경되더라도 프로그램의 수정이 불필요
- 멀티 태스킹을 위한 분할 처리를 하는 운영체제에서 하나의 응용프로그램이 출력장치를 독점하지 못하도록 운영체제를 통하여 장치에 간접접근



5

GDI와 디바이스 컨텍스트

□ DC (Device Context)

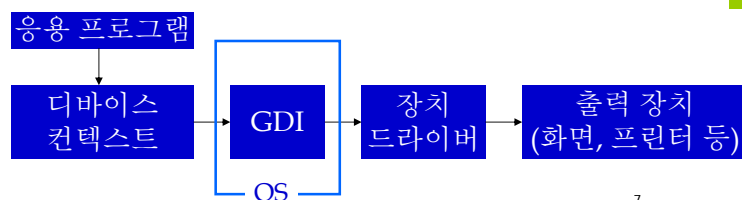
- **GDI가 생성하고 관리하는 데이터 구조체**
- DC를 통해 멀티태스킹(멀티스레딩) GUI 환경에서 발생할 수 있는 여러 상황을 고려하여 출력 가능
 - 예, 출력영역, 원점 정보, 그래픽 정보(글꼴, 펜, 브러시...)
- 기능
 - 응용프로그램의 출력에 대한 허가를 얻고, 그려지는 영역을 결정하는 역할
 - 윈도우의 클리핑 영역 (즉, 응용프로그램의 출력허용 영역) 관리
 - 그래픽 정보 관리 (펜, 브러시, 글꼴, 비트맵, 팔레트)
- DC를 얻고 나면, 반드시 작업 완료 시 해제 필요

6

GDI와 디바이스 컨텍스트

□ 윈도우 응용 프로그램의 출력 과정

- 출력하기 전에 윈도우 운영체제에게 **DC 요청**
- 운영체제의 GDI는 내부적으로 DC를 만든 후 DC 핸들(즉, HDC 타입, 32비트 포인터)을 해당 응용프로그램에게 넘겨줌
- 응용 프로그램에서는 받은 DC 핸들을 출력 관련 API 함수를 호출할 때 사용하고, 출력 관련 API 함수를 호출하면 GDI가 이를 처리
- GDI는 해당 응용 프로그램이 넘겨준 DC 핸들로 내부 데이터를 참조하여 다양한 상황을 고려해서 출력

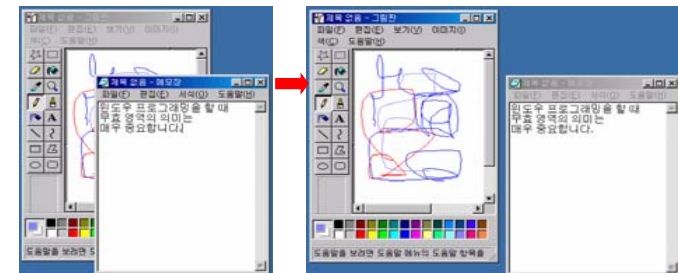


7

Invalid Region (무효 영역)

□ 메모장이 그림판의 일부를 가리고 있다가 가려진 부분이 드러나는 경우 (두가지 방법이 존재)

- 운영체제가 가려진 부분을 메모리에 임시 저장해 두었다가 다시 화면을 복구 - 운영체제가 화면 복구를 직접 담당하는 것
- 화면에 다시 그려야 할 부분을 프로그램에 알려주면 프로그램이 알아서 다시 그림 - 화면 복구를 각 응용프로그램이 담당하고 운영체제는 복구해야 할 시점과 복구할 영역정보 전달하는 것



Invalid Region (무효 영역)

- 화면을 다시 그려야 하는 상황
 - 무효영역을 다시 그리기 위해서 해당 응용프로그램으로 WM_PAINT 메시지를 보내어 처리될 수 있도록 함
 - WM_PAINT 메시지를 받으면 ::BeginPaint() 호출
 - 다시 그려야 하는 영역 (즉, 무효 영역)에 대한 정보 획득
 - 이 영역에 출력할 수 있는 HDC 획득
 - 화면 출력이 끝나면 ::EndPaint() 호출
- WM_PAINT 메시지 발생 상황
 - 윈도우가 생성될 때
 - 윈도우의 크기가 변경될 때
 - 최소화 또는 최대화 되었을 때
 - 다른 윈도우가 가렸다가 드러날 때

9

Invalid Region (무효 영역)

- 무효 영역 생성
 - 응용 프로그램 내에서 강제적으로 WM_PAINT 메시지를 발생시키기 위한 함수

배경을 지울 것인지?

// 전체영역 무효화

void CWnd::Invalidate (BOOL bErase = TRUE);

// 일부 사각영역만 무효화

void CWnd::InvalidateRect (LPCRECT lpRect, BOOL bErase = TRUE);

무효화 할 영역의 좌표

10

Invalid Region (무효 영역)

- WM_PAINT 메시지 처리
 - HelloSDK


```
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps); // DC를 얻음
    TextOut(hdc, 100, 100, str, lstrlen(str)); // 출력
    EndPaint(hwnd, &ps); // DC 반환
    return 0;
```
 - HelloMFC
 - DC를 얻고 반환하는 작업이 CDC 클래스의 생성자와 소멸자에서 수행

출력할 내용이 없어도 반드시 BeginPaint 를 호출할 것. 그렇지 않으면 OS가 계속 무효영역이 존재하는 것으로 간주함

```
void CMainFrame::OnPaint()
{
    char *msg = "Hello, MFC";
    CPaintDC dc(this);
    dc.TextOut(100, 100, msg);
}
```

MFC에서는 WM_PAINT 이벤트가 발생하면 OnPaint()가 수행됨.
내부적으로 BeginPaint/EndPaint 호출

11

출력 방법 비교

- SDK
 - 윈도우 운영체제에게 DC를 요청
 - 운영체제로부터 받은 DC 핸들을 사용하여 출력
 - 운영체제에게 DC 사용이 끝났음을 알림
- MFC
 - DC 객체 생성
 - DC 객체의 멤버 함수를 호출하여 출력

MFC의 CDC 파생 클래스를 이용하면 DC를 편리하게 다룰 수 있음

12

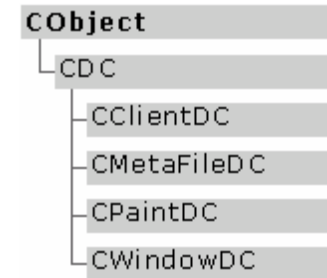
다양한 DC 클래스

클래스 이름	용도
CPaintDC	클라이언트 영역에 출력할 때 (WM_PAINT 메시지 핸들러에서만 사용)
CClientDC	클라이언트 영역에 출력할 때 (WM_PAINT 메시지 핸들러를 제외한 다른 모든 곳에서 사용)
CWindowDC	윈도우의 전체 영역(클라이언트 영역 + 비 클라이언트 영역)에 출력할 때
CMetaFileDC	메타 파일(Metafile)에 출력할 때

13

다양한 DC 클래스

□ 클래스 계층도



14

CPaintDC 클래스

- WM_PAINT 메시지가 발생했을 때 다시 그려져야 할 영역 (Invalid Rectangle)에 대한 DC 관리
- WM_PAINT 메시지 핸들러인 OnPaint() 함수에서 사용
- 내부적으로 BeginPaint() 함수를 호출하고 소멸자는 EndPaint() 함수를 호출
- CPaintDC 사용 예

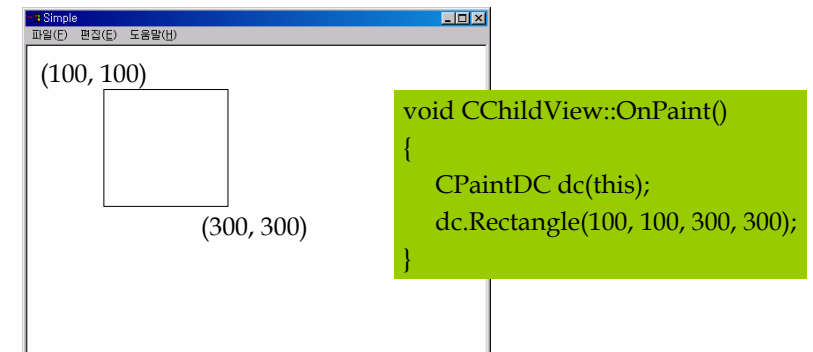
```

void CChildView::OnPaint()
{
    CPaintDC dc(this); // View의 client 영역에 출력할 수 있도록 DC 개체 생성
    // 출력 작업...
}
  
```

15

CPaintDC 클래스

- CPaintDC는 WM_PAINT 메시지가 발생할 때 마다 호출
 - 윈도우 크기가 변경하는 등의 상황에서도 항상 표시됨



16

CClientDC 클래스

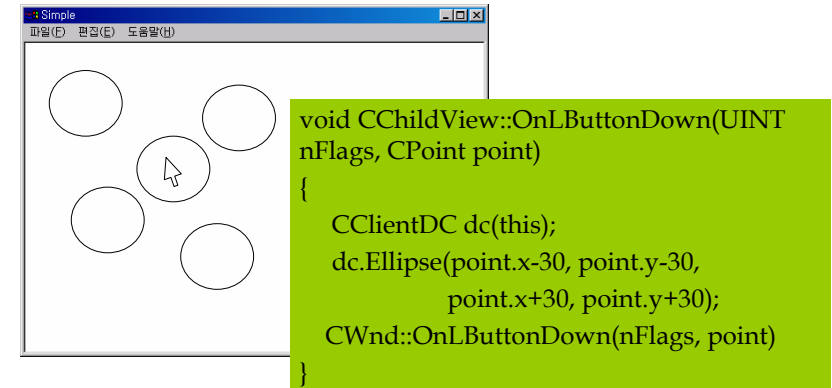
- 클라이언트 영역만을 표시하는 DC 관리
- 일시적으로 윈도우의 클라이언트 영역에서 그래픽 개체를 사용할 경우에 이용
- 생성자에서 GetDC 함수를 호출하여 DC를 얻고, 소멸자에서 ReleaseDC 함수를 호출하여 DC를 해제
- 생성시 출력 대상이 되는 윈도우의 핸들 값을 인자로 사용
- CClientDC 사용 예

```
void CChildView::OnLButtonDown(UINT nFlags, CPoint point)
{
    CClientDC dc(this);    // 현재 View 객체의 클라이언트 영역의 DC 생성
    // 출력 작업...
}
```

17

CClientDC 클래스

- 화면에 왼쪽 마우스 버튼을 누르면 그 점을 중심으로 반지름이 30인 원이 그려지는 예



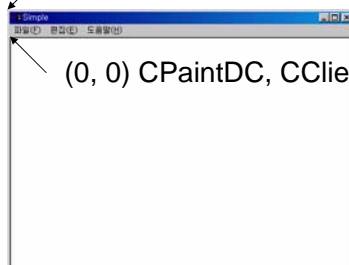
18

CWindowDC 클래스

- 윈도우 전체영역 (비클라이언트+클라이언트)에 그래픽 요소를 출력하고자 할 때 사용
- CWindowDC 사용 방법
 - CPaintDC, CClientDC 클래스와 동일
- 원점 위치

(0, 0) CWindowDC

(0, 0) CPaintDC, CClientDC



19

CMetaFileDC 클래스

- 메타 파일(Metafile)
 - GDI 명령어를 저장할 수 있는 파일
- CMetaFileDC 사용 방법
 - CMetaFileDC 객체를 만든 후 CMetaFileDC::Create() 호출
 - 메타 파일 객체를 일반적인 디바이스 컨텍스트 객체로 간주하고 출력 관련 멤버 함수를 호출
 - CMetaFileDC::Close()를 호출하면 윈도우 운영체제가 내부적으로 메타 파일을 만든 후 메타 파일 핸들(HMETAFILE 타입)을 리턴
 - CDC::PlayMetaFile()로 메타 파일을 실행

```
CMetaFileDC mdc;    // 메타 파일 DC 객체(순수한 C++ 객체) 생성
mdc.Create();        // 메타 파일 DC(운영체제가 인식하는) 생성
mdc.Ellipse(0, 50, 50, 100); // 메타 파일에 출력
mdc.Rectangle(50, 50, 100, 100); // 메타 파일에 출력
HMETAFILE m_hmf = mdc.Close(); // 메타 파일 핸들 저장
..
CClientDC dc(this);
dc.PlayMetaFile(m_hmf); // 메타 파일 실행
```

20

CDC 그래픽 함수

□ 점 찍기

이름	기능
GetPixel()	화면의 특정 위치에 해당하는 점의 색을 얻는다.
SetPixel()	화면의 특정 위치에 원하는 색의 점을 찍으며 원래의 점의 색을 리턴한다.
SetPixelV()	SetPixel과 기능은 동일하지만 SetPixel 함수와 달리 원래의 점의 색을 리턴하지 않으므로 속도가 더 빠르다.

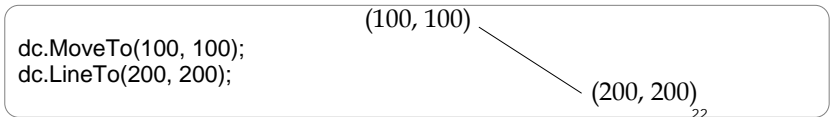
21

CDC 그래픽 함수

□ 선 그리기

- 한 번의 호출로 선을 그리는 함수는 제공하지 않으며 다음 두 단계를 거쳐야 함

이름	기능
MoveTo()	현재 위치를 옮긴다.
LineTo()	현재 위치로부터 특정 위치까지 선을 그린 후 현재 위치를 갱신한다.



CDC 그래픽 함수

□ 도형 그리기

이름	기능
Rectangle()	사각형을 그린다.
Ellipse()	사각형에 내접하는 타원을 그린다.
Polyline()	다각선을 그린다.
Polygon()	다각형을 그린다.
PolyBezier()	베지어 곡선을 그린다.

23

CDC 그래픽 함수

□ 텍스트 출력 함수

이름	기능
TextOut()	특정 위치에 문자열을 출력한다.
DrawText()	사각형을 기준으로 문자열을 출력한다.
SetTextColor()	문자의 색을 바꾼다.
SetBkColor()	문자의 배경색을 바꾼다.
SetTextAlign()	기준 위치에 대한 문자열의 정렬 방식을 정한다.

24

```

CString str;
int cx, cy, x1, x2, y1, y2;
CRect rect;
GetClientRect(&rect); //클라이언트 영역의 크기 얻기
cx = rect.Width() / 2; cy = rect.Height() / 2; //클라이언트 영역 중간 위치 얻기
str.Format("화면중간위치(%d:%d)", cx,cy);
pDC->TextOut(0, 0, str); //문자열 출력
//원 그리기 (클라이언트 영역에 내접하는 원)
pDC->Ellipse(rect);
//문자열 출력
pDC->DrawText(str, rect, DT_CENTER|DT_VCENTER|DT_SINGLELINE);
//선그리기, 화면높이의 중간위치에서 화면 폭의 중간위치까지 선 그리기
pDC->MoveTo(0, cy);
pDC->LineTo (cx, cy);

//다각형 그리기, 꼭지점의 좌표값 3개로 삼각형 그리기
CPoint ptData1[3]={CPoint(100,100), CPoint(150,50), CPoint(200,100)};
pDC->Polygon(ptData1, 3);

//베지어 곡선 그리기
CPoint ptData2[4]={CPoint(100,200), CPoint(150,100),
CPoint(200,300),CPoint(250,200)};
pDC->PolyBezier(ptData2, 4);

```

매핑 모드(Mapping Mode)

- 주어진 좌표가 화면상 실제 어디에 해당하는지 결정하는 방법
- 논리 좌표 (논리단위)
 - GDI 그래픽 함수들이 사용하는 모든 좌표
 - CDC에서 사용하는 윈도우 좌표
- 물리 좌표 (장치단위)
 - 실제 화면에 출력되는 좌표(픽셀 단위)
 - CWnd에서 사용하는 좌표
 - 뷰포트(viewport) : 물리 좌표가 사용되는 영역
- 매핑모드 관련 함수
 - SetMapMode(int fnMapMode);
 - int GetMapMode();
- 사용자 정의 매핑모드를 이용할 경우 범위지정 함수
 - SetWindowExt() : 논리단위의 범위지정함수
 - SetViewportExt() : 장치단위의 범위지정함수

26

매핑 모드 종류

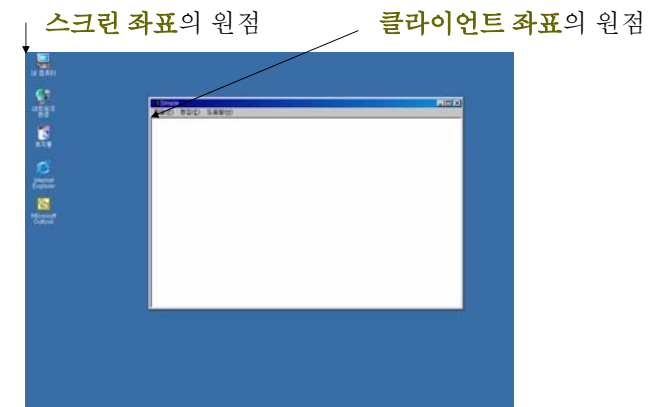
매핑 모드	단위	x축	y축
MM_TEXT	1 픽셀 (1:1로 mapping)	→ +x	↓ +y
MM_LOMETRIC	0.1 mm	→ +x	↓ -y
MM_HIMETRIC	0.01 mm	→ +x	↓ -y
MM_LOENGLISH	0.01 인치	→ +x	↓ -y
MM_HIENGLISH	0.001 인치	→ +x	↓ -y
MM_TWIPS	1/1440 인치	→ +x	↓ -y
MM_ISOTROPIC	사용자 정의 (가로, 세로 길이 동일)	사용자 정의	사용자 정의
MM_ANISOTROPIC	사용자 정의	사용자 정의	사용자 정의

SetMapMode(value)

27

물리좌표와 원점

- 스크린 좌표와 클라이언트 좌표

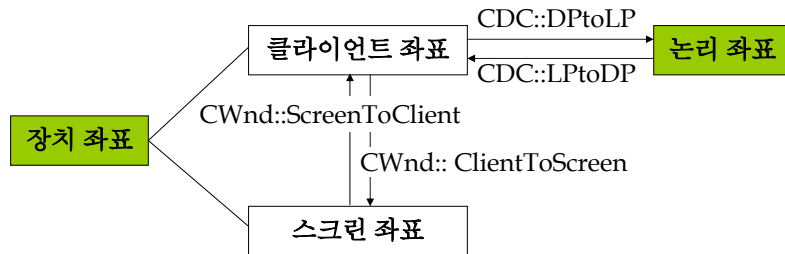


28

좌표 변환

좌표 변환 함수

- 논리 좌표 <-> 장치 좌표
 - void LPtoDP(LPPOINT lpPoints, int nCount = 1);
 - void DPtoLP(LPPOINT lpPoints, int nCount = 1);
- 클라이언트 좌표 <-> 스크린 좌표
 - void ScreenToClient(LPPOINT lpPoint);
 - void ClientToScreen(LPPOINT lpPoint);



29

DC의 속성 함수

속성	초기값	속성을 얻는 함수	속성을 변경하는 함수
텍스트 색상	검정색	GetTextColor()	SetTextColor()
배경 색상	흰색	GetBkColor()	SetBkColor()
배경 모드	OPAQUE	GetBkMode()	SetBkMode()
매핑 모드	MM_TEXT	GetMapMode()	SetMapMode()
그리기 모드	R2_COPYPEN	GetROP2()	SetROP2()
현재 위치	(0, 0)	GetCurrentPosition()	MoveTo()
펜	BLACK_PEN	SelectObject()	SelectObject()
브러시	WHITE_BRUSH	SelectObject()	SelectObject()
폰트	SYSTEM_FONT	SelectObject()	SelectObject()
비트맵	없음	SelectObject()	SelectObject()
팔레트	없음	SelectPalette()	SelectPalette()
영역	없음	SelectObject()	SelectObject()

이 속성에 따라 GDI 출력함수의 결과가 달라짐

30

그리기 모드 (DrawMode)

펜, 브러시에 적용되는 연산 - SetROP2(int nDrawMode)

- 그림을 그릴 때 사용하는 색상(S)와 스크린 원래 색상(D)의 혼합방법

그리기 모드	연산	그리기 모드	연산
R2_NOP	D = D	R2_NOT	D = ~D
R2_BLACK	D = BLACK	R2_WHITE	D = WHITE
R2_COPYPEN	D = S	R2_NOTCOPYPEN	D = ~S
R2_MERGEPEENNOT	D = ~D S	R2_MASKPENNOT	D = ~D & S
R2_MERGENOTPEN	D = ~S D	R2_MASKNOTPEN	D = ~S & D
R2_MERGEPEEN	D = D S	R2_NOTMERGEPEN	D = ~(D S)
R2_MASKPEN	D = D & S	R2_NOTMASKPEN	D = ~(D & S)
R2_XORPEN	D = S ^ D	R2_NOTXORPEN	D = ~(S ^ D)

바탕색 복원에 이용 (같은 그림을 두 번 그리기)

S (Source)
D (Destination)

//Raster Operation (DrawMode) 예

```

CClientDC dc(this);
CColorDialog dlgColor; //색상 다이얼로그의 인스턴스 생성
COLORREF penColor;
// 색상 다이얼로그에서 색상을 선택하고 확인을 누르면
if (dlgColor.DoModal() == IDOK) {
    penColor = dlgColor.GetColor(); // 선택한 색상 값을 얻어 PenColor에 설정
}
CPen pen, *oldpen;
pen.CreatePen(PS_SOLID, 2, penColor); // PEN 객체 생성 CPen(PenColor);
oldpen = dc.SelectObject(&pen); // PEN 객체 등록

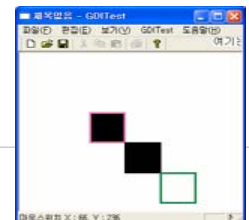
dc.SetROP2(R2_XORPEN); // XORPEN으로 설정 (보색으로 처리)
dc.Rectangle(100, 100, 150, 150);

dc.SetROP2(R2_BLACK); // BLACK으로 설정
dc.Rectangle(150, 150, 200, 200);

dc.SetROP2(R2_COPYPEN); // COPYPEN으로 설정
dc.Rectangle(200, 200, 250, 250);

dc.SelectObject(oldpen); // PEN 객체 복원
    
```

White color : RGB(255,255,255) Black color : RGB(0,0,0)



배경 모드 (BackgroundMode)

- 배경에 적용되는 연산 - SetBkMode(int nBkMode)
 - OPAQUE : 현재 배경색
 - Text를 출력하거나 도형을 그릴 때 현재 DC에 설정된 배경색으로 칠함
 - TRANSPARENT : 투명배경색
 - Text를 출력하거나 도형을 그릴 때 배경을 그대로 둠

```

CClientDC dc(this);
dc.SelectStockObject(LTGRAY_BRUSH); // 배경을 밝은 회색으로 칠한다.

Rect rect;
GetClientRect(&rect);
dc.Rectangle(&rect);           // 클라이언트 크기의 사각형을 출력.
dc.SetBkMode(TRANSPARENT); //투명배경으로 설정
dc.TextOut(100, 100, CString("투명배경 문자 모드")); //투명 배경인 문자열 출력

dc.SetBkMode(OPAQUE);         //기존 배경색으로 복원
dc.TextOut(100, 150, CString("불투명배경 문자 모드")); //초기 배경색으로 문자열 출력
    
```

33

GDI 객체

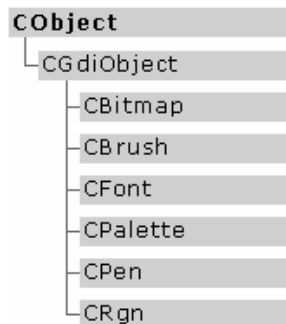
- GDI 객체
 - GDI에서 출력할 때 사용하는 도구
- 종류

GDI 객체	용도	클래스 이름
펜	선을 그릴 때	CPen
브러시	면의 내부를 채울 때	CBrush
폰트	문자를 출력할 때	CFont
비트맵	픽셀의 집합으로 이루어진 그림을 다룰 때	CBitmap
팔레트	출력될 색의 집합을 다룰 때	CPalette
영역	다양한 형태의 면을 정의할 때	CRgn

34

GDI 객체

- 클래스 계층도



MFC는 GDI 객체를 다룰 수 있도록 7개의 클래스를 제공함.
CGdiObject는 이들 클래스의 Base Class

35

GDI 객체

- GDI 객체 생성
 - GDI객체를 위한 각 클래스의 생성자나 Create함수 이용
 - CreatePen(), CreateSolidBrush(), CreateBitmap(), CreatePalette(), CreateFont(), CreatePolygonRgn()
- 객체를 Device Context에 등록
 - CDC::SelectObject() 함수 이용
 - 기존 설정된 객체는 반환되어 포인터로 저장
 - pOldGDIObj = dc.SelectObject(&newGDIObj)
- GDI 함수로 출력 작업
 - dc.Rectangle(..)
- Device Context에 이전 GDI객체로 환원
 - dc.SelectObject(pOldGDIObj)
- 객체 삭제
 - DeleteObject() 함수로 강제 삭제
 - GDI 객체가 소멸될 때 소멸자에 의한 자동 삭제

36

GDI 객체

□ CDC class의 SelectObject() - Afxwin.h 파일 참조

public:

```
virtual CGdiObject* SelectStockObject(int nIndex);
CPen* SelectObject(CPen* pPen);
CBrush* SelectObject(CBrush* pBrush);
virtual CFont* SelectObject(CFont* pFont);
CBitmap* SelectObject(CBitmap* pBitmap);
int SelectObject(CRgn* pRgn);
CGdiObject* SelectObject(CGdiObject* pObject);
```

37

GDI 객체 사용 예

```
...
CClientDC dc(this);           //CClientDC 생성
CPen myPen, *pOldPen; // 새로운 펜과 기존의 펜을 저장할 변수
myPen.CreatePen(PS_DOT, 1, RGB(0,0,255));
                               // 파란 점선을 갖는 펜 객체를 생성

pOldPen = dc.SelectObject(&myPen); // DC에 새로운 펜을 설정
dc.Rectangle(..);                 // 출력 작업(사각형이 파란점선으로 그려짐)

dc.SelectObject(pOldPen); // DC에 기존의 펜을 복원
```

38

펜

- 선이나 영역의 경계선을 그릴 때 사용
- 선의 두께, 색상, 선의 스타일 설정
- 생성 방법 - 생성자 (방법1) 또는 CreatePen (방법2) 이용

// 방법 1

```
CPen pen(PS_SOLID, 2, RGB(255, 0, 0));
```

// 방법2

```
CPen pen;
```

```
pen.CreatePen(PS_SOLID, 2, RGB(255, 0, 0));
```

□ 펜 스타일

PS_SOLID	_____
PS_DASH	- - - - -
PS_DOT
PS_DASHDOT	- . - . -
PS_DASHDOTDOT	- . . - .
PS_NULL	
PS_INSIDEFRAME	_____

39

펜

□ 사용 예 1

```
CPaintDC dc(this);           // DC 생성
CPen pen(PS_SOLID, 1, RGB(0, 0, 255)); // 펜 객체 생성
CPen *pOldPen = dc.SelectObject(&pen); // DC에 새로운 펜 객체 설정
dc.Rectangle(100, 100, 200, 200); // 사각형 그리기
dc.SelectObject(pOldPen);       // 이전 펜 객체 복원
```

□ 사용 예 2

- 실제로는 이전 펜의 주소를 저장하고 마지막으로 펜을 원상 복구하는 부분을 생략해도 무방함

```
CPaintDC dc(this);           // DC 생성
CPen pen(PS_SOLID, 1, RGB(0, 0, 255)); // 펜 객체 생성
dc.SelectObject(&pen);        // 펜 선택
dc.Rectangle(100, 100, 200, 200); // 사각형 그리기
```

40

브러시

- 영역의 내부를 채울 때 사용
- 영역에 채워질 색, 패턴 등을 설정
- 종류

브러시 종류	생성 예
솔리드 (Solid, 속이 채워짐)	<code>CBrush brush(RGB(255, 0, 0));</code> <code>CreateSolidBrush(RGB(255, 0, 0));</code>
해치 (Hatch, 교차된 평행선 무늬)	<code>CBrush brush(HS_DIAGCROSS, RGB(255, 0, 0));</code> <code>CreateHatchBrush(HS_DIAGCROSS, RGB(255, 0, 0));</code>
패턴 (Pattern, 비트맵의 반복 무늬)	<code>CBitmap bitmap;</code> <code>bitmap.LoadBitmap(IDB_BITMAP1);</code> <code>CBrush brush(&bitmap);</code>

41

브러시

- HatchBrush style

해치 브러시의 스타일	내용	모양
HS_BDIAGONAL	오른쪽에서 왼쪽으로 45도 내려가는 빗금	
HS_CROSS	십자가 형태의 빗금	
HS_DIAGCROSS	X자 형태의 빗금	
HS_FDIAGONAL	왼쪽에서 오른쪽으로 45도 내려가는 빗금	
HS_HORIZONTAL	수평으로 빗금	
HS_VERTICAL	수직으로 빗금	

42

브러시

- 사용 예 1

```
CPaintDC dc(this);
CBrush brush( RGB(255, 0, 0) );
CBrush *pOldBrush = dc.SelectObject(&brush);
dc.Ellipse(100, 100, 200, 200);
dc.SelectObject(pOldBrush);
```

- 사용 예 2

- 펜의 경우와 마찬가지로 실제로는 이전 브러시의 주소를 저장하고 마지막으로 다시 브러시를 원상 복구하는 부분을 생략해도 무방함

```
CPaintDC dc(this);
CBrush brush( RGB(255, 0, 0) );
dc.SelectObject(&brush);
dc.Ellipse(100, 100, 200, 200);
```

43

폰트 (Font)

- 문자출력을 위한 글자의 모양, 크기를 설정
- Font의 종류
 - 논리적인 폰트 : 이상적인 폰트에 대한 표현으로 실제로 존재하는 유사한 폰트를 얻기 위해 사용
 - 물리적인 폰트 : 실제로 시스템에 설치되어 있는 폰트로 실제 화면에 나타나는 폰트
- 폰트를 사용하는 방법
 - 폰트를 출력하기 위해서는 원하는 폰트에 대해 논리적인 폰트를 LOGFONT타입으로 기술하여 생성하고 DC에 폰트를 설정
 - 윈도우 GDI의 폰트 매퍼가 시스템에 설치되어 있는 폰트들 중에 가장 가까운 물리적인 폰트를 찾아 출력

44

Logical Font 구조체

Font(글꼴)에 대한 정보를 가지는 데이터 구조

```
typedef struct tagLOGFONT { /* if */
    LONG lfHeight;          // 문자높이
    LONG lfWidth;           // 문자폭
    LONG lfEscapement;       // 문자방향
    LONG lfOrientation;     // 문자회전각도
    LONG lfWeight;          // 문자굵기
    BYTE lfItalic;          // 기울임꼴
    BYTE lfUnderline;       // 밑줄
    BYTE lfStrikeOut;       // 취소선
    BYTE lfCharSet;         // 문자세트
    BYTE lfOutPrecision;    // output precision
    BYTE lfClipPrecision;   // clipping precision
    BYTE lfQuality;         // output quality
    BYTE lfPitchAndFamily;  // 문자간격과 글꼴패밀리
    TCHAR lfFaceName[LF_FACESIZE]; // 글자체 이름
} LOGFONT;
```

45

CFont 클래스

논리적 폰트를 생성하고 관리하는 클래스

새로운 글꼴 생성 메소드

- CFont::CreatePointFont(..)
 - 폰트의 크기 (포인트 단위)와 이름만으로 폰트를 생성, 나머지는 기본값으로 설정
- CFont::CreateFont(...)
 - LOGFONT 구조체가 가지는 모든 값들을 직접 매개변수로 사용하여 폰트 생성
- CFont::CreateFontIndirect(...)
 - LOGFONT 구조체 포인터를 매개변수로 사용하여 폰트 생성

```
CFont font;          // CFont 객체 생성
font.CreateFont(...); // argument가 16개로 너무 복잡
// font.CreateFontIndirect(...); // CFont 객체에 대해 Create*() 함수 호출
// font.CreatePointFont(400, Arial); // 400/10 포인트 크기 Arial 폰트
// font.CreatePointFontIndirect(...);
```

46

Font 공통 다이얼로그 박스

Font 공통 다이얼로그 박스의 LOGFONT구조체에 저장된 값으로 논리적 폰트 객체 생성

```
public:
    LOGFONT m_logFont; //logical font 구조체 변수 선언
....
// 폰트 공통 다이얼로그 박스로부터 logical font 구조체 얻어옴
CFontDialog dlg(&m_logFont);
if (dlg.DoModal() == IDOK) {
    dlg.GetCurrentFont(&m_logFont);
    // 폰트 공통 다이얼로그 박스에서 선택된 논리적 폰트정보 얻기
    str.Format("글 자체=%s, 크기=%d", dlg.GetFaceName(),
               dlg.GetSize()/10);
    // 글자체와 크기(포인트 단위의 10배로 얻어짐) 출력
}
```

47

TEXTMETRIC 구조체

```
typedef struct tagTEXTMETRIC { /* tm */
    int tmHeight;
    int tmAscent;
    int tmDescent;
    int tmInternalLeading;
    int tmExternalLeading;
    int tmAveCharWidth;
    int tmMaxCharWidth;
    int tmWeight;
    BYTE tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
} TEXTMETRIC;
```

- DC에 선택된 논리적 폰트와 관련된 물리적 폰트의 특징을 나타냄
- CDC::GetTextMetrics(LPTEXTMETRIC lpMetrics)으로 얻어옴

48

```

CFont newFont, *pOldFont=NULL;
TEXTMETRIC tm;
CString str;
CClientDC dc(this);
//폰트 공통 다이얼로그 박스로부터 logical font 구조체 얻어옴
CFontDialog dlg(&m_logFont);
if (dlg.DoModal() == IDOK) {
    dlg.GetCurrentFont(&m_logFont);
    str.Format("글자체=%s, 크기=%d", dlg.GetFaceName(), dlg.GetSize()/10);
}
// logical font 구조체를 이용한 폰트객체 생성
if (newFont.CreateFontIndirect(&m_logFont)) {
    pOldFont = dc.SelectObject(&newFont); //DC에 새로운 폰트객체 설정
    dc.TextOut(10,50, str);
    dc.GetTextMetrics(&tm); //DC의 폰트정보 얻어옴
    str.Format("글자크기(폭=%d, 높이=%d)", tm.tmAveCharWidth, tm.tmHeight);
    dc.TextOut(10,100, str);
    newFont.DeleteObject();
    newFont.CreatePointFont(200, "Arial"); //크기와 글꼴만으로 간단하게 폰트생성, 20point
    dc.SelectObject(&newFont);
    str.Format(" PointFont(200, Arial)");
    dc.TextOut(10, 150, str);
    if (pOldFont)
        dc.SelectObject(pOldFont); // 원래 font 복원
}

```

Stock GDI Object (내장 객체)

이름	용도
BLACK_PEN	폭이 1 픽셀인 검정색 펜
WHITE_PEN	폭이 1 픽셀인 흰색 펜
NULL_PEN	투명 펜
BLACK_BRUSH	검정색 브러시
DKGRAY_BRUSH	어두운 회색 브러시
GRAY_BRUSH	회색 브러시
LTGRAY_BRUSH	밝은 회색 브러시
HOLLOW_BRUSH 또는 NULL_BRUSH	투명 브러시
SYSTEM_FONT	윈도우 운영체제가 사용하는 폰트 예) 메뉴, 대화상자, ...

- 윈도우 운영체제가 미리 만들어서 제공하는 GDI 객체
 - 내장 객체는 생성과정을 생략하고 CDC::SelectStockObject() 함수를 사용하여 DC에 선택한다.

Stock GDI Object (내장 객체)

□ 사용 예

```

CClientDC dc(this); // DC 생성
CBrush* pOld;
pOld = (CBrush*)dc.SelectStockObject(GRAY_BRUSH);
// 내장된 회색 브러시를 선택
... // 실제 출력 작업
dc.SelectObject(pOld);

```

비트맵

- 복잡한 그림을 처리할 때 많이 사용
- 비트맵 정보

```

int CBitmap::GetBitmap (BITMAP* pBitMap);

typedef struct tagBITMAP {
    int bmType;
    int bmWidth; // 비트맵의 폭(픽셀 단위)
    int bmHeight; // 비트맵의 높이(픽셀 단위)
    int bmWidthBytes;
    BYTE bmPlanes;
    BYTE bmBitsPixel;
    LPVOID bmBits;
} BITMAP;

```

비트맵

□ 비트맵 정보 출력

- LoadBitmap(UINT nIDResource) 리소스 ID로 비트맵 얻기
- GetBitmap(BITMAP* pBitMap) 비트맵 정보 구조체 얻기

```
CBitmap bitmap;
bitmap.LoadBitmap(IDB_BITMAP1);
BITMAP bmpinfo;
bitmap.GetBitmap(&bmpinfo);
TRACE("가로 = %d, 세로 = %d\n", bmpinfo.bmWidth, bmpinfo.bmHeight);
```

53

비트맵

□ 비트맵은 곧바로 화면에 출력하는 함수를 제공하지 않음

□ 비트맵 출력 절차

- CBitmap::LoadBitmap() 함수를 이용하여 비트맵 리소스로부터 읽어오기
- CDC::CreateCompatibleDC() 함수를 이용하여 메모리 DC를 생성
- CDC::SelectObject() 함수를 이용하여 비트맵을 메모리 DC에 선택
- CDC::BitBlt() 또는 CDC::StretchBlt() 함수를 이용하여 화면에 출력
- CDC::SelectObject() 함수를 이용하여 DC를 복원

54

Memory Device Context

- 메모리의 일부를 마치 화면처럼 다룰 수 있도록 운영체제에서 제공하는 개념
- 응용프로그램에서는 메모리 DC에 비트맵을 선택 후 실제 물리적인 출력 장치로 내보냄
 - BitBlt 또는 StretchBlt 함수에서 Blt는 Block Transfer의 약자로서 이는 비트맵 데이터를 메모리에서 메모리로 고속 전송한다는 의미

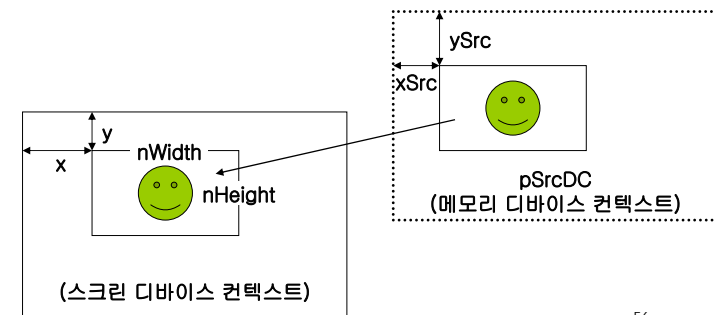
55

비트맵

□ 비트맵 출력 함수 - BitBlt

- 메모리 DC에서 화면 DC로 비트맵 블록을 그대로 전송하는 함수

```
BOOL BitBlt (int x, int y, int nWidth, int nHeight, CDC* pSrcDC,
             int xSrc, int ySrc, DWORD dwRop) ;
```



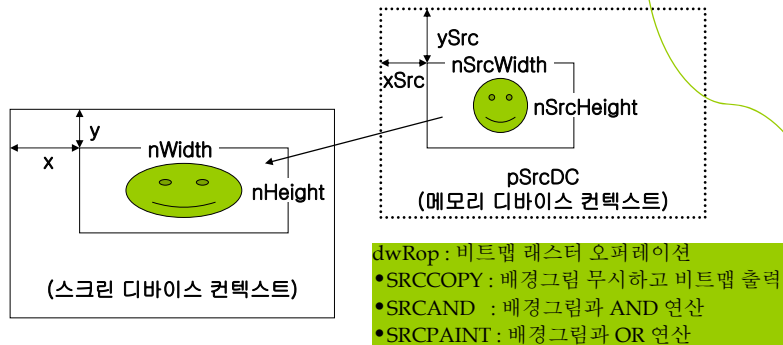
56

비트맵

□ 비트맵 출력 함수 - StretchBlt

- 비트맵을 확대하거나 축소하여 비트 블럭을 전송

```
BOOL StretchBlt (int x, int y, int nWidth, int nHeight, CDC* pSrcDC, int
xSrc, int ySrc, int nSrcWidth, int nSrcHeight, DWORD dwRop) ;
```



//Bitmap을 리소스에서 읽어서 화면에 출력하는 예

```
CPaintDC dc(this);
```

// 비트맵 리소스를 로드한 후 크기 정보를 얻는다.

```
CBitmap bitmap;
```

```
bitmap.LoadBitmap(IDB_BITMAP1); //비트맵 리소스 읽어오기
```

```
BITMAP bmpinfo;
```

```
bitmap.GetBitmap(&bmpinfo); //비트맵 정보 얻기(크기 등)
```

// 메모리 디바이스 컨텍스트를 만든 후 비트맵을 선택해 넣는다.

```
CDC dcmem;
```

```
dcmem.CreateCompatibleDC(&dc);
```

```
dcmem.SelectObject(&bitmap);
```

// 비트맵을 화면에 출력한다.

```
dc.BitBlt(10, 10, bmpinfo.bmWidth, bmpinfo.bmHeight,
```

```
&dcmem, 0, 0, SRCCOPY); //비트맵의 원래크기 유지
```

```
dc.StretchBlt(100, 100, bmpinfo.bmWidth*4, bmpinfo.bmHeight*6,
```

```
&dcmem, 0, 0, bmpinfo.bmWidth, bmpinfo.bmHeight, SRCCOPY);
```

//비트맵의 크기를 변형

58

//Bitmap을 브러시로 받아서 타원 안에 출력하는 예

```
CClientDC dc(this);
```

// 비트맵 리소스를 로드한 후 크기 정보를 얻는다.

```
CBitmap bitmap2;
```

```
bitmap2.LoadBitmap(IDB_BITMAP2); //비트맵 리소스 읽어오기
```

```
CBrush bitmapBrush(&bitmap2);
```

```
dc.SelectObject(&bitmapBrush);
```

// 비트맵을 타원 안에 출력한다.

```
dc.Ellipse(800, 30, 110, 330);
```

// 이미지 파일 Bitmap을 읽어 들어서 화면에 출력하는 예

```
CClientDC dc(this);
```

// 비트맵 이미지 파일을 로드한 후 BITMAP으로 핸들을 얻는다.

```
HBITMAP hImage = (HBITMAP) LoadImage(AfxGetInstanceHandle(),
```

```
"멀티미디어.bmp",
```

```
IMAGE_BITMAP, 0, 0, LR_LOADFROMFILE);
```

// 메모리 디바이스 컨텍스트를 만든 후 비트맵을 선택해 넣는다.

```
CDC dcmem;
```

```
dcmem.CreateCompatibleDC(&dc);
```

```
dcmem.SelectObject(&hImage);
```

// 비트맵을 화면에 출력한다.

```
dc.StretchBlt(100, 100, 777, 272, &dcmem, 0, 0, 777, 272, SRCCOPY);
```

60

Practice

- SimpleDrawing 프로그램을 이용하여 선, 사각형, 타원, 비트맵 등 그리기를 구현하라
 - 먼저 상단 중앙에 본인의 이름을 쓴다
 - DrawText(..) 또는 TextOut(...)
 - 내장객체를 이용하여 경계선이 검정색이고 내부가 비어있는 타원과 경계선이 없고 내부가 회색인 타원 그리기
 - SelectStockObject(...)
 - Ellipse(...)
 - 4개의 선 (blue dashdot style)으로 사각형 그리기
 - CPen bdpn(...)
 - MoveTo(...) & LineTo(...)
 - 비트맵 패턴(본인이 원하는 비트맵사용가능)을 넣은 사각형 그리기
 - CBrush bitmapBrush(&bitmap)
 - Rectangle(...)