オブジェクト指向とゲームプログラミング

▶ DirectInput編 － 第３回　デバイスオブジェクトの生成 ◀

DirectInputDevice8オブジェクトの生成と初期化

DirectInputDevice8オブジェクトは、入力デバイスを制御するためのオブジェクトです。このオブジェクトは、キーボード、マウス、ゲームパッドといった入力デバイスから情報を取得するメソッドを提供しており、入力デバイスごとに生成する必要があります。
DirectInputDevice8オブジェクトは、以下の手順で生成および初期化します。

①DirectInputDevice8オブジェクトの生成とインタフェースの取得
②データフォーマットの設定
③協調レベルの設定
④プロパティの設定
⑤アクセス権の取得

①デバイスオブジェクトの生成
始めに、入力デバイスを制御するためのDirectInputDevice8オブジェクト(デバイスオブジェクト)を生成します。このオブジェクトは、IDirectInput8::CreateDeviceメソッドで生成します。

IDirectInput8::CreateDeviceメソッド
－説明－
CreateDeviceメソッドは、指定された入力デバイスのGUIDに基づき、DirectInputDevice8オブジェクトを生成し、そのインタフェースを返します。

－書式－
```
HRESULT CreateDevice(REFGUID rguid, LPDIRECTINPUTDEVICE* lplpDirectInputDevice,
                     LPUNKNOWN pUnkOuter);
```

－パラメータ－
１つ目の引数(rguid)は、入力デバイスのGUID(デバイスごとに個別に割り当てられた128ビットの値)です。この値はEnumDevicesメソッドでデバイスを列挙することにより取得できます。キーボードとマウスは、あらかじめ定義された以下の値を指定できます。

GUID_SysKeyboard	デフォルトのキーボード
GUID_SysMouse	デフォルトのマウス

２つ目の引数(lplpDirectInputDevice)は、生成されるオブジェクトのインタフェースを格納する変数のアドレスです。LPDIRECTINPUTDEVICE8型またはIDirectInputDevice8*型の変数のアドレスを指定します。

３つ目の引数(pUnkOuter)は、通常は使用しないのでNULLにします。

－戻り値－
成功した場合はDI_OK、それ以外はエラーコードを返します。

```
// キーボードのデバイスオブジェクトを生成(pDInputは初期化済みのDirectInput8オブジェクト)
IDirectInputDevice8*   pDIDevice;   // デバイスオブジェクトへのインタフェース
pDInput->CreateDevice(GUID_SysKeyboard, &pDIDevice, NULL);
```

②データフォーマットの設定
デバイスオブジェクトが生成できたら、データフォーマットの設定を行います。データフォーマットとは、入力デバイスからの情報をどのような形式で受け取るかを定義するもので、IDirectInputDevice8::SetDataFormatメソッドで設定します。
SetDataFormatメソッドの引数は、データ形式を設定したDIDATAFORMAT構造体型変数のアドレスです。この構造体を設定することにより、アプリケーションにあわせた情報を入力デバイスから取得することができます。キーボード、マウス、ゲームパッドに関しては、あらかじめ定義されている以下の変数を指定することができます。

変　数	対応デバイス	GetDeviceStateメソッドで返される情報
c_dfDIKeyboard	キーボード	256バイトの配列
c_dfDIMouse	マウス	DIMOUSESTATE構造体

c_dfDIMouse2	マウス	DIMOUSESTATE2構造体
c_dfDIJoystick	ゲームパッド	DIJOYSTATE構造体
c_dfDIJoystick2	ゲームパッド (フォースフィードバック)	DIJOYSTATE2構造体

設定したデータ形式は、GetDeviceStateメソッドでデバイスの情報を取得したときに返されます。

```
// データフォーマットの設定(キーボード)
pDIDevice->SetDataFormat(&c_dfDIKeyboard);
```

③協調レベルの設定

DirectInputでは、入力デバイスごとに協調レベルを設定する必要があります。協調レベルとは、デバイスをほかのアプリケーションとどのように協調して使用するかを示すものです。

協調レベルの設定は、IDirectInputDevice8::SetCooperativeLevelメソッドで行います。このメソッドの１つ目の引数はメインウィンドウのハンドル、２つ目の引数は協調レベルを表すフラグで、以下の２つのモードを組み合わせて指定します。

①「DISCL_FOREGROUND」(フォアグランド)または「DISCL_BACKGROUND」(バックグランド)
フォアグランドモードでは、アプリケーションが非アクティブ状態になると自動的にデバイスの入力を停止します。バックグランドモードでは常に入力を続けます。

②「DISCL_EXCLUSIVE」(排他占有)または「DISCL_NONEXCLUSIVE」(非排他)
排他モードでは、入力デバイスのアクセス権を解放しない限り、ほかのアプリケーションがその入力デバイスを制御することができなくなります。非排他モードでは、ほかのアプリケーションの妨げになることはありません。
排他占有すると、アクセス権を解放するまでDirectInputでしか制御できなくなり、協調レベルを設定した以外のウィンドウはAPIを使っても入力を受け取ることができなくなります。キーボードやマウスを排他占有すると、それらに関するメッセージが発生しなくなり、メッセージボックスのOKボタンをクリックできなくなったりします。最悪の場合はOSを終了できなくなります。ほとんどの入力デバイスは非排他モードで十分ですが、ゲームパッドなどのフォースフィードバック機能を制御するときは、排他占有モードに設定します。

```
// 協調レベルの設定(フォアグランド、非排他モードに設定。hWndはメインウィンドウのハンドル)
pDIDevice8->SetCooperativeLevel(hWnd, DISCL_FOREGROUND | DISCL_NONEXCLUSIVE);
```

④プロパティの設定

必要なら、デバイスバッファの設定や、マウスおよびゲームパッドの軸モードといった入力デバイスの動作属性(プロパティ)を、IDirectInputDevice8::SetPropertyメソッドで設定します。

⑤アクセス権の取得

デバイスの設定がすべて正しく終えたら、IDirectInputDevice8::Acquireメソッドを呼び出し、アクセス権を取得します。アクセス権を取得すると、入力デバイスから情報を取得できるようになります。

```
// アクセス権取得
pDIDevice->Acquire();
```

デバイスオブジェクトの解放

デバイスオブジェクトの解放は、ほかのオブジェクトと同じようにReleaseメソッドで行いますが、その前にIDirectInputDevice8::Unacquireメソッドでアクセス権を解放しておきます。

```
pDIDevice->Unacquire();      // アクセス権解放
pDIDevice->Release();        // デバイスオブジェクト解放
```

課　　題

CDInput8クラスに、デバイスオブジェクトの生成と解放を行う機能を追加しましょう。

(１)キーボードやマウスといった入力デバイスをカプセル化したクラスを作成します。

DirectInputDeviceオブジェクトは、IDirectInputDevice8インタフェースをとおして操作を行います。このインタフェースは、指定された入力デバイスが無い場合やデバイスオブジェクトの生成に失敗した場合、NULLを保持します。ゲームプログラムでは、一部の入力デバイスが無くても支障ありません。たとえば、キーボードかゲームパッドのいずれかがあれば問題ない、という状況が挙げられます。つまり、必要な入力デバイスがすべて生成失敗とならない限り、プログラムをそのまま実行し続けて構わないということになります。

この場合、IDirectInputDevice8インタフェースがNULLの場合とそうでない場合が考えられます。NULLは具体的なインスタンスを参照しておらず、そもそも読むことも書くこともできない領域を指しているので、NULLを介してメソッドを呼び出すと、Windowsは例外を発生させてプログラムを強制終了します。例外が発生しないようにするためには、そのインタフェースがNULLかどうかを調べなければなりません。

```
// pDIDeviceは、IDirectInputDevice8*型の変数(DirectInputDeviceのインタフェース)
if(pDIDevice != NULL) {
    // pDIDeviceを使った処理(状態取得など)
}
```

しかし、プログラムによってはこのようなチェックが非常に多くなってしまい、コードが煩雑になってしまう危険性があります。そこで、デザインパターンのNull Object(Null Device)パターンを適用します。
Null Objectパターンは、インタフェースやポインタがインスタンスを参照していないことを表現するのに、NULLを用いるのではなく、明示的に「何もしないオブジェクト」(Null Object)を使うようにするのです。そのNull Objectは、呼び出される可能性のあるメソッドはすべて持ちますが、すべて「何もしない」処理として実装します。このようなNull Objectを導入すると、NULLチェックを省くことができ、簡潔なコードになります。

Null Objectパターンを適用した入力デバイスクラスのヘッダファイル(InputDevice.hpp)を以下のように作成しましょう。

－InputDevice.hpp－

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
    Microsoft Windows2000/XP

  【コンパイラ】
    Microsoft Visual C++ 2005

  【プログラム】
    InputDevice.hpp
                入力デバイスクラスヘッダ

  【履歴】
    * Version    1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

#pragma once

/*********************************************************************************************/
/*                                        バージョン宣言                                     */
/*********************************************************************************************/
#define DIRECTINPUT_VERSION 0x0800

/*********************************************************************************************/
/*                                      インクルードファイル                                 */
```

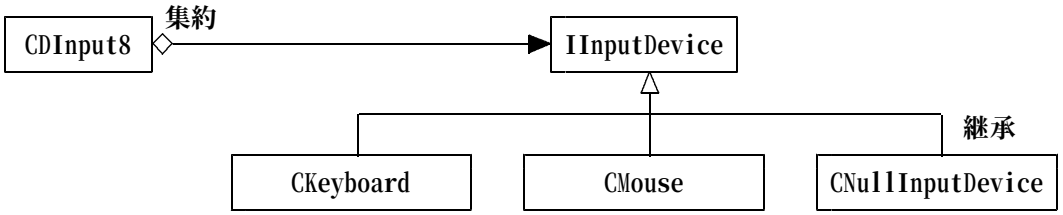
```
/*****************************************************************************************************/
#include <dinput.h>

/*****************************************************************************************************/
/*                                    入力デバイスインタフェース定義                                   */
/*****************************************************************************************************/
class IInputDevice {
public:
    virtual ~IInputDevice() {}

    virtual bool IsNull() const = 0;
};

/*****************************************************************************************************/
/*                                       キーボードクラス定義                                        */
/*****************************************************************************************************/
class CKeyboard : public IInputDevice {
public:
    CKeyboard(IDirectInputDevice8* pDIDKeyboard);
    virtual ~CKeyboard();

    virtual bool IsNull() const { return ※ここは各自考えましょう; }

private:
    IDirectInputDevice8*   m_pDIDKeyboard;
};

/*****************************************************************************************************/
/*                                        マウスクラス定義                                           */
/*****************************************************************************************************/
class CMouse : public IInputDevice {
public:
    CMouse(IDirectInputDevice8* pDIDMouse);
    virtual ~CMouse();

    virtual bool IsNull() const { return ※ここは各自考えましょう; }

private:
    IDirectInputDevice8*   m_pDIDMouse;
};

/*****************************************************************************************************/
/*                                    NULL入力デバイスクラス定義                                     */
/*****************************************************************************************************/
class CNullInputDevice : public IInputDevice {
public:
    CNullInputDevice() {}
    virtual ~CNullInputDevice() {}

    virtual bool IsNull() const { return ※ここは各自考えましょう; }
};
```

IInputDeviceクラスは、CKeyboardクラス、CMouseクラス、CNullInputDeviceクラスのインタフェースとして使用したいので、それらのクラスの基底クラスにし、publicな関数は基本的に純粋仮想関数として宣言します。CDInput8クラスがキーボードやマウスのデバイスオブジェクトを生成しようとしたとき、その成否によって、CKeyboardクラスまたはCMouseクラスのオブジェクトか、CNullInputDeviceオブジェクトを生成(するように設計)します。アップキャストにより、IInputDeviceへのポインタには、そのすべてを代入することができます。IInputDeviceポインタをとおせば、そのインスタンスがCKeyboard, CMouse, CNullInputDeviceのいずれのオブジェクトでも、それを意識することなく操作することができるというわけです。

しかし、このままでは、そのオブジェクトがNull Objectなのかどうかがわからなくなってしまいます。そこで、Null ObjectかどうかをIsNull関数を各クラスに実装します。この関数は、Null ObjectならTrue, そうでなければFalseを返すように実装します。

CKeyboardクラスがキーボードをカプセル化したクラス、CMouseクラスがマウスをカプセル化したクラスになります。CNullInputDeviceクラスがNull Object役になります。今後、CNullInputDeviceクラスには、IInputDeviceクラスのpublicな関数をすべて「何もしない処理」として実装します。

最終的なクラス相関図は、以下のようになります。



(２)入力デバイスクラスのソースファイル(InputDevice.cpp)を以下のように作成しましょう。

－InputDevice.cpp－

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
     Microsoft Windows2000/XP

  【コンパイラ】
     Microsoft Visual C++ 2005

  【プログラム】
     InputDevice.cpp
                入力デバイスクラス

  【履歴】
     * Version     1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

/*************************************************************************************************/
/*                                        インクルードファイル                                   */
/*************************************************************************************************/
#include "InputDevice.hpp"
#include <cassert>

/*************************************************************************************************/
/*                                        コンストラクタ                                         */
/*************************************************************************************************/
CKeyboard::CKeyboard(IDirectInputDevice8* pDIDKeyboard)
{
    assert(pDIDKeyboard != NULL);
    m_pDIDKeyboard = pDIDKeyboard;
    m_pDIDKeyboard->AddRef();
}

/*************************************************************************************************/
/*                                        デストラクタ                                           */
/*************************************************************************************************/
CKeyboard::~CKeyboard()
{
    m_pDIDKeyboard->※ここは各自考えましょう;
    m_pDIDKeyboard->Release();
}

/*************************************************************************************************/
/*                                        コンストラクタ                                         */
/*************************************************************************************************/
CMouse::CMouse(IDirectInputDevice8* pDIDMouse)
{
    assert(pDIDMouse != NULL);
    m_pDIDMouse = pDIDMouse;
    m_pDIDMouse->AddRef();
}
```

```
/*****************************************************************************************************/
/*                                          デストラクタ                                             */
/*****************************************************************************************************/
CMouse::~CMouse()
{
    m_pDIDMouse->※ここは各自考えましょう;
    m_pDIDMouse->Release();
}
```

CKeyboardクラスとCMouseクラスのコンストラクタでは、渡されたインタフェースのAddRefメソッドを呼び出し、参照カウンタを増やしています。COMのインタフェースを別の変数にコピーした場合、その期間が一時的でない場合は、このようにした方が安全です。このプログラムの場合は、CKeyboardオブジェクトまたはCMouseオブジェクトが存在している間は、各インタフェースが参照しているオブジェクトが解放されなくなるからです。もちろん、このままでは参照カウンタが増えたままになり解放されなくなるので、オブジェクトがデストラクトされるときに、Releaseメソッドで参照カウンタを減らします。

(３)CDInput8クラスに、以下のメンバを追加しましょう。

```
    IInputDevice*   m_pKeyboard;    // キーボード
    IInputDevice*   m_pMouse;       // マウス
```

(４)CDInput8クラスの適切な場所に、キーボードとマウスを制御するためのインタフェースを取得する以下の関数を追加しましょう。

```
    IInputDevice* GetKeyboard() const { return m_pKeyboard; }
    IInputDevice* GetMouse()    const { return m_pMouse;    }
```

(５)CDInput8クラスのコンストラクタに、以下の初期化子を追加しましょう。

```
    m_pKeyboard(NULL), m_pMouse(NULL)
```

(６)CDInput8::Initialize関数の適切な場所に以下のプログラムを追加し、入力デバイスが初期化されるまでNull Objectを保持するようにしましょう。

```
    m_pKeyboard = new CNullInputDevice();       // キーボードにNULLオブジェクトを設定
    m_pMouse    = new CNullInputDevice();       // マウスにNULLオブジェクトを設定
```

(７)CDInput8::Release関数の適切な場所に以下のプログラムを追加し、DirectInput解放時に入力デバイスの資源も解放されるようにしましょう。

```
    delete m_pMouse;       m_pMouse    = NULL;    // マウス解放
    delete m_pKeyboard;    m_pKeyboard = NULL;    // キーボード解放
```

(８)CDInput8クラスのprivateメンバに、入力デバイスを解放してNull Objectを設定する以下のReleaseDevice関数を追加しましょう。

```
/*****************************************************************************************************/
/*                                          デバイス解放                                             */
/*****************************************************************************************************/
void CDInput8::ReleaseDevice(IInputDevice*& pDevice)
{
    // pDeviceがNULLまたはNULLオブジェクトの場合は解放しない
    if(pDevice == NULL || pDevice->IsNull())
        return;

    // デバイスを解放し、NULLオブジェクトを設定
    delete pDevice;
    pDevice = new CNullInputDevice();
}
```

(９)CDInput8クラスの適切な場所に、キーボードとマウスを解放する以下のプログラムを追加しましょう。

```
    void ReleaseKeyboard() { ReleaseDevice(m_pKeyboard); }
    void ReleaseMouse()    { ReleaseDevice(m_pMouse);    }
```

(10)CDInput8クラスの適切な場所に、キーボードのデバイスオブジェクトを生成、初期化する以下のCreateKeyboard関数の足りない部分を補い、追加しましょう。

```
/*****************************************************************************************************/
/*                                          キーボードデバイス生成                                   */
/*****************************************************************************************************/
bool CDInput8::CreateKeyboard()
{
    if(m_pDInput == NULL) {
        ::OutputDebugString("*** Error - DirectInput未初期化(CDInput8_CreateKeyboard)\n");
        return false;
    }

    // キーボードオブジェクトが生成済みの場合は再生成しない
    if(m_pKeyboard->IsNull() == false)
        return true;

    // デバイス生成
    IDirectInputDevice8*   pDIDevice = NULL;
    if(m_pDInput->???????????(GUID_SysKeyboard, &pDIDevice, NULL) != DI_OK) {
        ::OutputDebugString("*** Error - デバイス生成失敗(CDInput8_CreateKeyboard)\n");
        return false;
    }

    // データフォーマット設定
    if(pDIDevice->???????????(&c_dfDIKeyboard) != DI_OK) {
        ::OutputDebugString("*** Error - データフォーマット設定失敗(CDInput8_CreateKeyboard)\n");
        pDIDevice->Release();
        return false;
    }

    // 協調レベル設定
    if(pDIDevice->??????????????????(m_hWnd, DISCL_FOREGROUND | DISCL_NONEXCLUSIVE) != DI_OK) {
        ::OutputDebugString("*** Error - 協調レベル設定失敗(CDInput8_CreateKeyboard)\n");
        pDIDevice->Release();
        return false;
    }

    // アクセス権取得
    pDIDevice->???????();

    // キーボードオブジェクト生成
    delete m_pKeyboard;
    m_pKeyboard = new CKeyboard(pDIDevice);
    pDIDevice->Release();

    return true;
}
```

(11)CDInput8クラスの適切な場所に、マウスのデバイスオブジェクトを生成、初期化する以下のCreateMouse関数の足りない部分を補い、追加しましょう。

```
/*****************************************************************************************************/
/*                                          マウスデバイス生成                                       */
/*****************************************************************************************************/
bool CDInput8::CreateMouse()
{
    if(m_pDInput == NULL) {
        ::OutputDebugString("*** Error - DirectInput未初期化(CDInput8_CreateMouse)\n");
        return false;
    }

    // マウスオブジェクトが生成済みの場合は再生成しない
    if(m_pMouse->IsNull() == false)
        return true;

    // デバイス生成
    IDirectInputDevice8*   pDIDevice = NULL;
```

```
    if(※ここは各自考えましょう != DI_OK) {
        ::OutputDebugString("*** Error - デバイス生成失敗(CDInput8_CreateMouse)\n");
        return false;
    }

    // データフォーマット設定
    if(※ここは各自考えましょう != DI_OK) {
        ::OutputDebugString("*** Error - データフォーマット設定失敗(CDInput8_CreateMouse)\n");
        pDIDevice->Release();
        return false;
    }

    // 協調レベル設定
    if(pDIDevice->??????????????????(m_hWnd, DISCL_FOREGROUND | DISCL_NONEXCLUSIVE) != DI_OK) {
        ::OutputDebugString("*** Error - 協調レベル設定失敗(CDInput8_CreateMouse)\n");
        pDIDevice->Release();
        return false;
    }

    // アクセス権取得
    pDIDevice->※ここは各自考えましょう;

    // マウスオブジェクト生成
    delete m_pMouse;
    m_pMouse = new CMouse(pDIDevice);
    pDIDevice->Release();

    return true;
}
```

(12)CDInput8.cppの適切な場所に、以下のプログラムを追加しましょう。

```
    #pragma comment(lib, "dinput8.lib")
```

静的リンクライブラリ「dinput8.lib」は、「c_dfDIKeyboard」などの定義済みの変数を使用するために必要となります。

(13)キーボードを使用する場合、以下のプログラムをアプリケーションを初期化する関数の適切な場所に追加しましょう。

```
    // キーボード生成
    if(DInput().CreateKeyboard() == false)
        return false;
```

(14)マウスを使用する場合、以下のプログラムをアプリケーションを初期化する関数の適切な場所に追加しましょう。

```
    // マウス生成
    if(DInput().CreateMouse() == false)
        return false;
```

キーボードまたはマウスのいずれかがあればプログラム続行可能な場合は、以下のようにします。

```
    if((DInput().CreateKeyboard() & DInput().CreateMouse()) == false)
        return false;
```

(15)キーボードとマウスにアクセスするためには「DInput().GetKeyboard()->メンバ関数名」のような長い名前が必要になります。以下のようなインライン関数を定義しておくと、「DIKeyboard()->メンバ関数名」というわかりやすく、かつ短い名前で使用することができるようになります。

```
    inline IInputDevice* DIKeyboard() { return DInput().GetKeyboard(); }
    inline IInputDevice* DIMouse()    { return DInput().GetMouse();    }
```