Chapter-6 Singletonパターン

6.1【ハンズオン】インスタンスの生成を一個に制限する

　では、サンプルプログラムのSingleton_01を利用して、インスタンスの生成の抑制のみの機能を持っているクラスを実装してみましょう。
　今回はゲームのプログラムでよく実装される、インゲームのマネージャー的なクラスのインスタンスの生成を一個に制限してみましょう。ちなみに、インゲームとはゲーム業界で使われている用語で、メインのゲームプレイシーンのことを指しています。例えば、対戦格闘ゲームであれば、キャラクター同士が格闘をしている、コアとなる部分のことです。

step-1 唯一のインスタンスを記憶するためのstaticメンバ変数を宣言する。

　まずは、InGameクラスにインスタンスのアドレスを記憶するためのstaticメンバ変数を宣言します。InGame.hの該当するコメントの箇所に次のプログラムを入力してください。
[InGame.h]

```
// step-1 唯一のインスタンスを記憶するためのstaticメンバ変数を宣言する。
static InGame* m_instance;  // 唯一のインスタンス。
```

step-2 staticメンバ変数を定義する。

　続いて、先ほど宣言したstaticメンバ変数の定義を追加します。InGame.cppに次のプログラムを入力してください。
[InGame.cpp]

```
// step-2 staticメンバ変数を定義する。
// C++は宣言ダメではNGで定義も用意する必要がある。
// インスタンスの生成抑制のためには、nullptrで初期化するのが重要。
InGame* InGame::m_instance = nullptr;
```

step-3 インスタンスの生成を抑制する。

　このステップがインスタンスの生成の抑制のトリックになります。InGame.cppに次のプログラムを入力してください。
[InGame.cpp]

```
// step-3 インスタンスの生成を抑制する。
if (m_instance != nullptr) {
  // m_instanceの値がnullptrではないということは、
  // すでにインスタンスが作成されているということになる。
  MessageBox(nullptr, L"InGameクラスのインスタンスが複数作られています！", L"エラー", MB_OK);
  std::abort();
}
```

```
// インスタンスのアドレスを記憶する。
m_instance = this;
```

このプログラムがInGameクラスのコンストラクタに実装されている点に注目してください。
InGameクラスのコンストラクタでは、一番最初にm_instanceが記憶しているアドレスをチェックしています。そして、もし記憶しているアドレスがnullptrではない場合はエラーメッセージを表示して、ランタイムエラーを発生させています。m_instanceがnullptrではないということは、過去にInGameクラスのインスタンスを作成したことがあることになります。
この実装によって、インゲーム中に、何かの間違いでInGameクラスのインスタンスが複数作られてしまうと、プログラムがクラッシュします。このトリックにより、プログラムの論理エラーに、ランタイムにはなりますが気づくことができるようになっています。

step-4 インスタンスのアドレスの記憶をクリアする。

次はデストラクタを利用したトリックです。InGameクラスのインスタンスは複数同時に存在するのがNGなだけで、複数回作ることは許可したい場合がほとんどです。
例えば格闘ゲームのような、タイトルシーン→キャラセレクトシーン→インゲームシーン→リザルトシーンという遷移をするゲームを考えてみてください。このような場合、InGameクラスを作るのは、インゲームシーンになるのですが、ほとんどの格闘ゲームでインゲームの後、リザルトシーンに遷移し、キャラクタセレクトに戻ることができると思います。ですので、インゲームクラスのインスタンスは、インゲームとキャラクターセレクトのループで、生成と破棄を繰り返すことになります。
しかし、step-3の実装だけでは、InGameクラスのインスタンスを複数回作成することはできません。そこで、InGameクラスのインスタンスが破棄されたら、m_instanceが記憶しているアドレスもクリアしてやるようにしましょう。InGameクラスのデストラクタに次のプログラムを入力してください。
[InGame.cpp]

```
// step-4 インスタンスのアドレスの記憶をクリアする。
// インスタンスが破棄されたので、m_instanceにnullptrを代入する。
m_instance = nullptr;
```

これでInGameクラスの実装は完了です。

step-5 InGameクラスのインスタンスを生成する。

ここからは、作成したInGameクラスを利用していましょう。まずはInGameクラスのインスタンスを生成します。main.cppの該当するコメントの箇所に次のプログラムを入力してください。
[main.cpp]

```
// step-5 InGameクラスのインスタンスを生成する。
InGame inGame;
```

step-6 InGameクラスのインスタンスを生成する。

続いて、InGameクラスのUpdate()関数とDraw()関数を呼び出します。今回のインゲームクラスの仕事は、InGame::Update()関数の中でPlayerの更新処理と描画処理、InGame::Draw()関数の中で地面の描画処理を呼び

出すことです。
[main.cpp]

```
// step-6 インゲームの更新処理と描画処理を実行する。
inGame.Update();
inGame.Draw( renderContext );
```

step-7 InGameクラスのインスタンスを生成する。

　では、これで最後です。最後のステップは論理エラーを起こすプログラムになります。つまりプログラムの間違いです。本来InGameクラスのインスタンスは同時に一つしか存在することができません。しかし、プログラマーの間違いで、InGameクラスのインスタンスを複数生成してしまったわけです。では、main.cppに次のプログラムを入力してください。
[main.cpp]

```
// step-7 ゲームパッドの入力でInGameクラスのインスタンスを新たに生成する。
if (g_pad[0]->IsTrigger(enButtonA) ){
    InGame* inGame2 = new InGame;
}
```

　入力出来たら実行してみてください。実行すると地面の上に女の子が立っているだけのプログラムが実行されます。ここでコントローラのAボタンを押してみて下さい。うまく実装できていると、エラー通知のメッセージボックスが表示されてゲームがクラッシュします。