Ｃ＋＋ワンポイントレッスン

シングルトンデザインパターンについて

ソフトウエア研究会

シングルトンデザインパターン

シングルトンデザインパターンとは、システムで唯一となるオブジェクトを提供するデザインパターン。最も簡単なのは、グローバルな変数としてクラスオブジェクトを定義することで実現できる。

例:

```
class Hoge {
…
} hoge;
```

上記の例では、Hogeというクラスの実体hogeがグローバル変数として生成される。ユーザーは、プログラムのすべての位置からhogeを参照し使用することが可能となる。

シングルトンデザインパターンを使うとき

- デバイスの個数に由来する場合。たとえば、マウス入力を扱うクラスなど。（マウスを2個同時に使うアプリケーションは稀なため）
- システム由来で唯一である必要があるもの。たとえば、スレッドを管理するスレッドマネージャーなど。
- アプリケーション本体など、二重に動作すると困るもの。
- インスタンスへのアクセスが容易になる。システムの広範囲から呼ぶようなロギング機能などで便利。

以上、おおむね、アプリケーションプログラムの中には数個のシングルトンオブジェクトが存在している。

シングルトンの実装方法

1. グローバル変数として定義。(Javaではできない)
2. すべてのメソッドとインスタンスをstaticで宣言する
3. アクセス用のstaticメソッドを用意その１(new/delete)

```
hoge* getInstance() {
    if (m_instance == 0) m_instance = new Hoge;
    return m_instance;
}
```

4. アクセス用のstaticメソッドを用意その２(staticなローカル変数)

```
hoge* getInstance() {
    static Hoge instance;
    return &instance;
}
```

※いずれの場合も、コンストラクタをprivateにして、他者からの生成を抑制すること。

シングルトンの考察と問題点1

- グローバルオブジェクトとシングルトンの違い
- シングルトンの唯一性を保障するためのよりよい方法
- シングルトンの破棄と破棄後に行われたアクセスの検出
- シングルトンオブジェクトにおける進んだ寿命管理の実装
- マルチスレッドの問題

シングルトンの考察と問題点２

- グローバル変数やすべてstaticなクラスは、プログラムがロードされた時に生成され、exit命令後に破棄されるため、生成と破棄の順番をユーザーが知るすべがない。そのため、シングルトンオブジェクトから別のシングルトンオブジェクトを呼び出す場合、呼び出し先のオブジェクトが生成前もしくは破棄後の可能性があり、安全にアクセスできない。
- getInstance等のstaticなメソッドで生成する場合は、最初に呼ばれたときにインスタンスが生成されるため、生成の順序はユーザーが管理できる。特に、メソッド内でのstaticなローカル変数と定義した場合、コンパイラが自動的にオブジェクトの生成と破棄を行うため、便利である。

前頁のstaticなローカル変数の実現例 （緑色の文字はコンパイラが自動的に生成するコード)

```
hoge* getInstance() {
  static bool _initialized = false;
  static char _buffer[sizeof(Hoge)];
  _threadLock();
  if (!_initialized) {
    _CostructHoge(_buffer); // Hogeのコンストラクタを呼び出す
    atexit(_DestroyHoge);  // Hogeのデストラクタを呼び出す関数を登録する
    _initialized = true;
  }
  _threadUnLock();
  return *reinterpret_cast<Hoge*>(_buffer);
  static Hoge instance;
  return &instance;
}
```

シングルトンまとめ

- シングルトンの実装は、staticメソッド内のstaticなローカル変数による実装がよい。（スコットメイヤーズのシングルトン）
- その場合でも、破棄の順序は管理できないので、シングルトンオブジェクトのデストラクタから他のシングルトンオブジェクトのアクセスは避けなければならない。
- シングルトンデザインパターンはオブジェクト指向とは相反する面があるため、設計段階で使用を吟味するべきである。