Utilizando classes Singleton em Symbian OS

Sigleton é um padrão de projeto definido no livro *Design Patterns: Elements of Reusable Object-Oriented Software*. Esse padrão diz que para uma determinada classe apenas uma única instância deve existir por todo o programa.

Em C++ padrão a forma mais simples de se criar uma classe `Singleton` é utilizar um ponteiro estático e um método estático. Em Symbian OS isto funciona para EXE mas não para DLL*. Pois este ponteiro estático é considerado como um *Writable Static Data* (variável global que existe durante toda a vida do processo), e isto não é permitido para DLLs.

Para fazer a criação de singletons temos duas possibilidades, utilizar a classe base `CCoeStatic`, ou utilizar *Thread Local Storage* (TLS). Até o momento nunca utilizei TLS então irei focar na utilização da classe `CCoeStatic`. A principal desvantagem na utilização de `CCoeStatic` é que você não pode utilizá-lo em aplicações baseadas em console (sem interface gráfica).

A lógica utilizada será bem parecida com o singleton em C++ padrão, a diferença é que não iremos utilizar um ponteiro estático. Iremos precisar criar um UID único para a classe (único dentro da aplicação), ele será utilizado para identificar de qual classe iremos obter a instância, caso esteja trabalhando com várias classes `Singleton`.

ExemploSingleton.h

```
/*
* Exemplo de criação de classes Singleton utilizando CCoeStatic.
* Autor: s60brasil.blogspot.com
*/

#include // Cabeçalho de CCoeStatic.

class CExemploSingleton : public CCoeStatic
{
    public:
       /* Retorna uma instância desta classe.
        * Na primeira vez que este método for chamado ele fará a criação da  instância.
        * Em chamadas subsequentes será retornada esta instância.
        */
       static CExemploSingleton* InstanceL();
       ~CExemploSingleton();
    private:
       // Construtor precisa ser privado para evitar que outras classes a instancie
diretamente.
       CExemploSingleton();
       // Construtor de segunda-fase caso precise realizar alguma operação que
possa abandonar
       void ConstructL();
};
```

ExemploSingleton.cpp

```
#include "ExemploSingleton.h"

/* UID definido para a classe CExemploSingleton.
 * Este UID precisa ser único para esta classe dentro da aplicação.
 */
const TUid KUidClasseSingleton = { 1 };

/* Construção da classe.
 * Passamos para CCoeStatic a UID da classe e prioridade de destruição da instância.
```


(C) Copyright Nokia 2014. All rights reserved.

```
 */
CExemploSingleton::CExemploSingleton() : CCoeStatic( KUidClasseSingleton,
EDefaultDestructionPriority )
{

}

CExemploSingleton::~CExemploSingleton()
{
     // Não implementado.
}

CExemploSingleton* CExemploSingleton::InstanceL()
{
     CExemploSingleton* instancia = static_cast
                 ( CCoeEnv::Static( KUidClasseSingleton ) );
     /* Somente entrará neste if na primeira chamada a InstanceL();
      * Chamadas subsequentes a este método retornarâo esta instância.
      */
     if( !instancia )
     {
         instancia = new (ELeave) CExemploSingleton();
         /* Adiciona o ponteiro ao CleanupStack para garatir que está referência  não
será perdida caso ConstructL abandone.*/
         CleanupStack::PushL( instancia );
         // Chamada ao construtor de segunda-fase
         instacia->ConstructL();
         // Após chamar métodos que possam abandonar , liberar da pilha
         CleanupStack::Pop( instancia );
     }

     return instancia;
}

void CExemploSingleton::ConstructL();
{
     // Não implementado
}
```

O código por si só é bastante fácil de compreender. Como disse anteriormente basta fazer chamadas ao método `InstanceL()` para receber a referência da classe `Singleton`. Na primeira chamada será alocado o ponteiro e nas outras será apenas retornado a instância. Algo que vale ressaltar é a construção da classe:

```
CExemploSingleton::CExemploSingleton() : CCoeStatic( KUidClasseSingleton,
EDefaultDestructionPriority )
{

}
```

Passamos através do contrutor para `CCoeStatic` dois argumentos. Uma das definições do construtor de `CCoeStatic` é:

```
CCoeStatic(TUid aUid, TInt aDestructionPriority, TScope aScope=EThread);
```

Onde:

(C) Copyright Nokia 2014. All rights reserved.

aUid = Uid da classe singleton.
aDestructionPriority = Define com que prioridade a instância será destruída. Quanto maior este valor , mais cedo o objeto será destruído. Valores positivos fazem com que o objeto seja destruído antes de `CCoeAppUi` enquanto valores negativos fazem com que o objeto seja destruído após `CCoeAppUi`.
aScope = Define para quem o objeto estará acessível. Por definição é para a *thread* da aplicação.

(*) = A partir da versão 8.1, passou a ser possível utilizar dessa maneira.

Referência

Series60 Brasil Blog
SymbianOS SDK

(C) Copyright Nokia 2014. All rights reserved.