

```

/*****
*          utils.h
*
* Thu Nov 22 12:00:01 2007
* Copyright 2007 Ceacy
* ceacy@free.fr
*****/

/*
(Entête GNU)
*/
#include <stdexcept>
#include <gmpxx.h>

typedef unsigned int uint;
typedef mpf_class mfloat;

#ifndef UTILS_INC_DONE
#define UTILS_INC_DONE
    namespace Utils
    {

        template<class T> class Mat8x8
        {
            T m_mat[8][8];
        public:
            // Indexés de 1 à 8
            T& operator() (uint i, uint j)
            {
                if( !i || i > 8 || !j || j > 8 )
                {
                    throw new std::out_of_range( "Les indices doivent être compris
entre 1 et 8" );
                }
                else
                {
                    return m_mat[i-1][j-1];
                }
            }
            T operator() (uint i, uint j) const
            {
                if( !i || i > 8 || !j || j > 8 )
                {
                    throw new std::out_of_range( "Les indices doivent être compris
entre 1 et 8" );
                }
                else
                {
                    return m_mat[i-1][j-1];
                }
            }
        };
    }

    Base class for design pattern "singleton".
    Note : this class doesn't allow singleton to have parameters in their

```

constructor.
an "init()"

To have the same effect, provide a static boolean "mInitialized", and
function : if mInitialized is false, call init().

```
*/
template<class T> class Singleton
{
public:
    /* Access the instance */
    static T& getSingleton();
    static T* getSingletonPtr();
    /* Delete the instance */
    static void freeSingleton();
protected:
    Singleton() {};
    virtual ~Singleton() {};
private:
    static T* mInstance;
    /* No copy, please. */
    Singleton( Singleton& );
    void operator =( Singleton& );
};
template<class T> T* Singleton<T>::mInstance = 0;
template<class T> T& Singleton<T>::getSingleton()
{
    if( !mInstance )
    {
        mInstance = new T();
    }
    return *mInstance;
}
template<class T> T* Singleton<T>::getSingletonPtr()
{
    if( !mInstance )
    {
        mInstance = new T();
    }
    return mInstance;
}
template<class T> void Singleton<T>::freeSingleton()
{
    if( mInstance )
    {
        delete mInstance;
        mInstance = 0;
    }
}
}

#endif
```