10/31/2012

PIMPL

Alexander Seitel

Division of Medical and Biological Informatics (MBI)
German Cancer Research Center Heidelberg

dkfz.
GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Easy example

Declaration of a class representing a student:

```
#ifndef STUDENT_HEADER
#define STUDENT_HEADER

#include <string>

class Student
{
private:
std::string m_name;
};

#endif
```

Assume, that many classes include this header

Easy example

What happens during compile time when these lines are added?

```
#ifndef STUDENT_HEADER
#define STUDENT_HEADER

#include <string>
#include "ContactNo.h"

class Student
{
private:
std::string m_name;
ContactNo m_contactNo;
};

#endif
```

→ Every class including Student is compiled again

How can this be avoided?

Using the **P**rivate **IMPL**ementation pattern.

```
#ifndef STUDENT_HEADER
#define STUDENT_HEADER

#include <string>

class Student
{
private:
std::string m_name;
struct PIMPL; //Forward declaration.
PIMPL* pimpl; //The opaque pointer.

public:
Student();
~Student();
};

#endif
```

```
#include "Student.h"
#include "ContactNo.h"

//Private Implementation structure.
struct Student::PIMPL
{
ContactNo m_contactNo;
};

Student::Student()
{
pimpl = new PIMPL;
}

Student::~Student()
{
if(pimpl)
delete pimpl;
}
```

Pack private implementation of the class into a simple internal structure

Make opaque- or **d-pointer** to this structure private member of this class

Another example (from MITK)

```
#ifdef MITK_USE_TOF_KINECT
#include <XnCppWrapper.h>
#endif
  class KinectController : public itk::Object
  {
  public:
    mitkClassMacro( KinectController , itk::Object );
    itkNewMacro( Self );
  protected:
    KinectController();
    ~KinectController();
  private:
    ErrorText();
#ifdef MITK_USE_TOF_KINECT
    xn::Context m_Context; ///< OpenNI context
#endif
  };
```

Only applicable if external library (OpenNI) is available

→ m_Context must also be „`#ifdef`ed“ in whole .cpp file

Solution: PIMPL

```
#ifdef MITK_USE_TOF_KINECT
#include <XnCppWrapper.h>
#endif
  class KinectController : public itk::Object
  {
  public:
    mitkClassMacro( KinectController , itk::Object );
    itkNewMacro( Self );
  protected:
    KinectController();
    ~KinectController();
  private:
    ErrorText();
    class KinectControllerPrivate;
#ifdef MITK_USE_TOF_KINECT
    KinectControllerPrivate *d;
    xn::Context m_Context; ///< OpenNI context
#endif
  };
```

Pack private implementation of the class into a simple internal class

Solution: PIMPL

```
#include "mitkKinectController.h"
#include <XnCppWrapper.h>

class KinectController::KinectControllerPrivate
{
public:

KinectControllerPrivate();
~KinectControllerPrivate();

bool ErrorText(unsigned int error);
xn::Context m_Context; ///< OpenNI context
};

KinectController::KinectControllerPrivate::KinectControllerPrivate():  m_Context(NULL){}

KinectController::KinectControllerPrivate::~KinectControllerPrivate(){}

bool KinectController::KinectControllerPrivate::ErrorText(unsigned int error)
{
return true;
}

KinectController::KinectController(): d(new KinectControllerPrivate){}

KinectController::~KinectController()
{
delete d;
}
```

→ Header now independent of external library

→ no „`#ifdef`ing“ in code. .cpp stub can e.g. be used if library unavailable

PIMPL at a glance

Pros:

- no recompile when changing private members
- no #include for classes used „by value“ in private members
- Hide internal implementation details from the user

→ Reduction of compile time

→ Especially benefitial for big projects

Cons:

- more complex code
- dynamic allocation required for every instance of the class
- does not work for protected member used by subclasses

Thank you!

Questions?

References

- The PIMPL idiom http://c2.com/cgi/wiki?PimplIdiom
- The PIMPL pattern http://angelorohit.blogspot.de/2008/02/reducing-compile-times-in-large-c.html