Cheshire Cat C++ idiom

Topics

1. What is Cheshire cat idiom?
2. Concepts behind the Cheshire cat idiom
3. Merits & Demerits of using Cheshire cat idiom
4. Good practices
5. An example with all good practices.

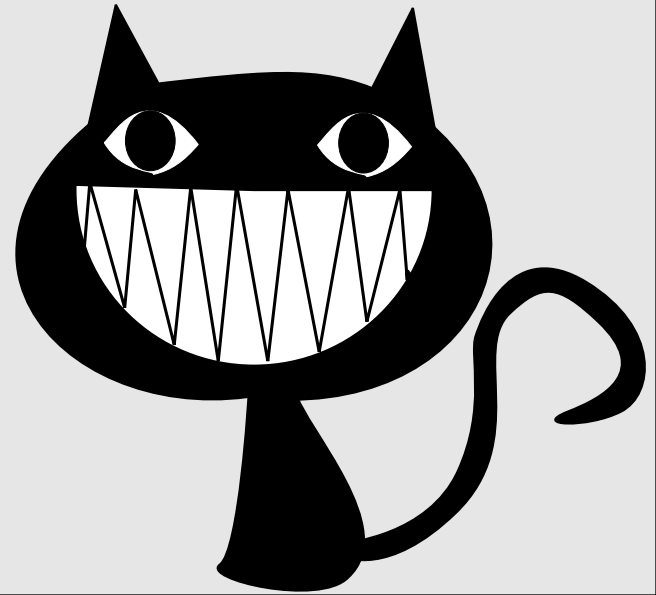
Cheshire Cat ????

- The Cheshire Cat is a fictional cat in movie “Alice in Wonderland”

Cheshire Cat ????

- The Cheshire Cat is a fictional cat in movie “Alice in Wonderland”

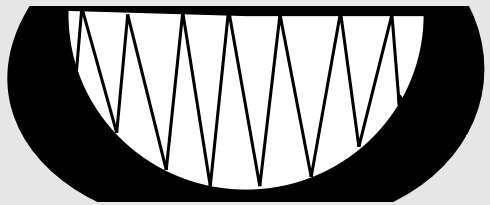
- One of the distinguishing feature of Cheshire cat is that,
  - From time to time its **body disappears**.



Cheshire Cat ????

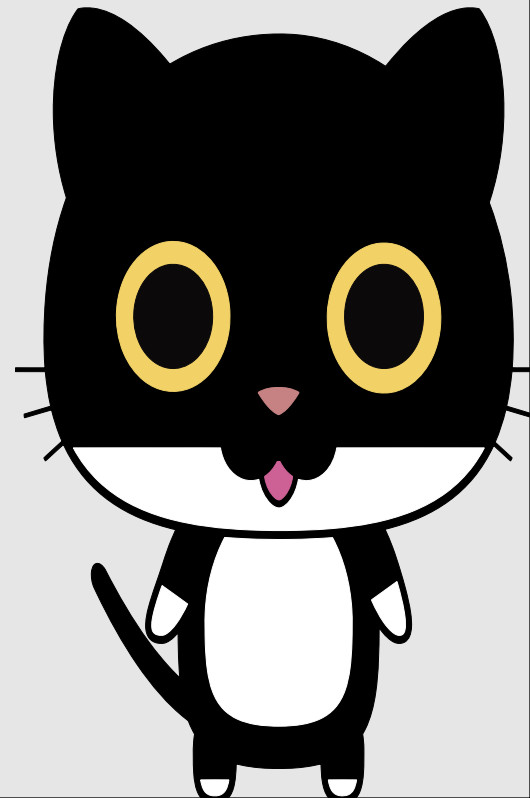
- The Cheshire Cat is a fictional cat in movie “Alice in Wonderland”

- One of the distinguishing feature of Cheshire cat is that,
  - From time to time its **body disappears**.
  - The last thing visible is its **iconic grin.**



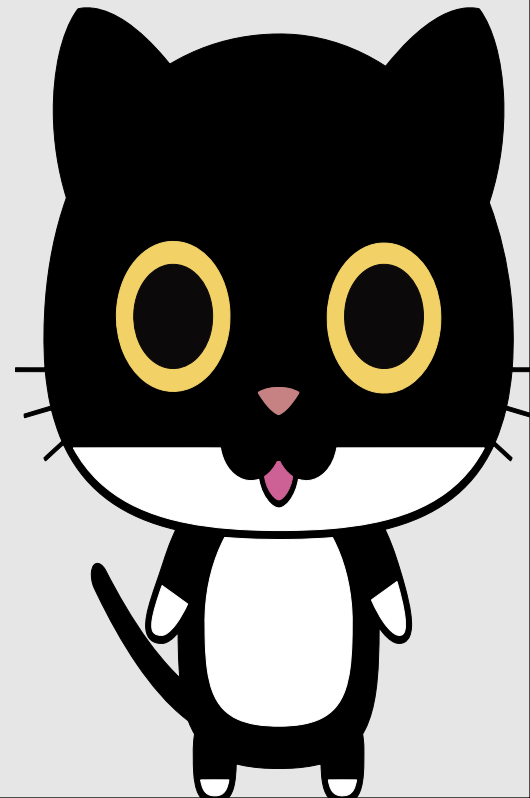
Cheshire Cat idiom in C++

- The idiom is also well known as



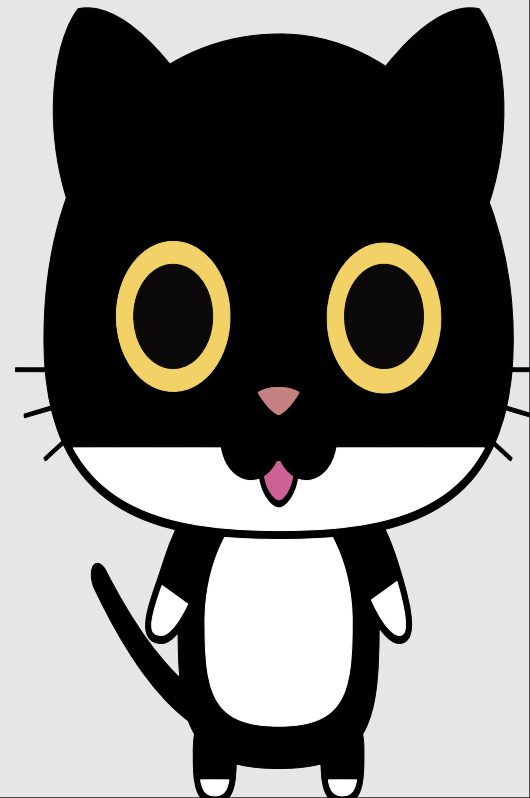
Cheshire Cat idiom in C++

- The idiom is also well known as
  - Compiler firewall idiom



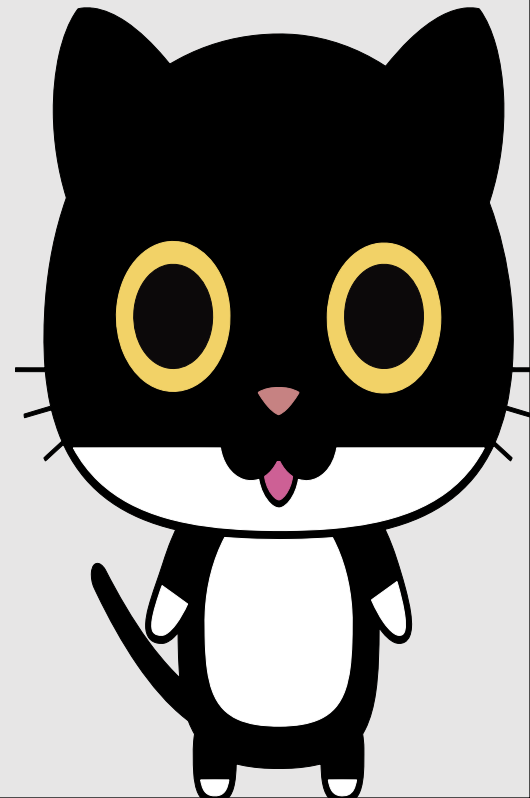
Cheshire Cat idiom in C++

- The idiom is also well known as
  - Compiler firewall idiom
  - d-pointer



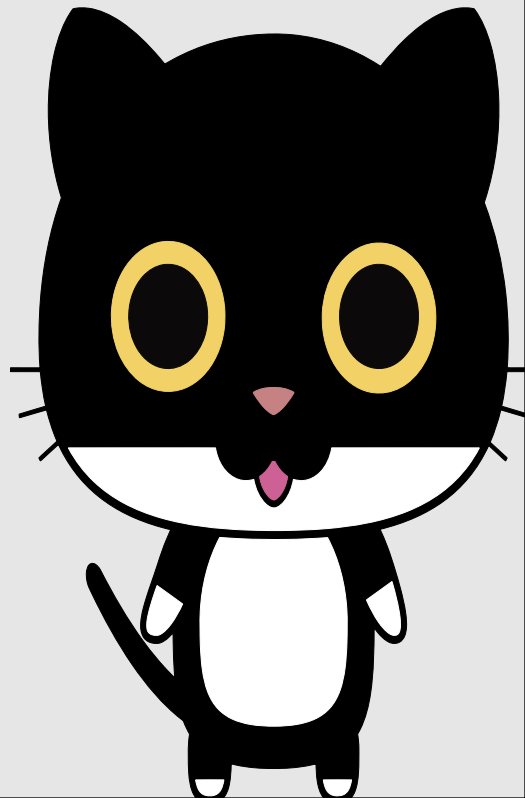
Cheshire Cat idiom in C++

- The idiom is also well known as
  - Compiler firewall idiom
  - d-pointer
  - handle classes



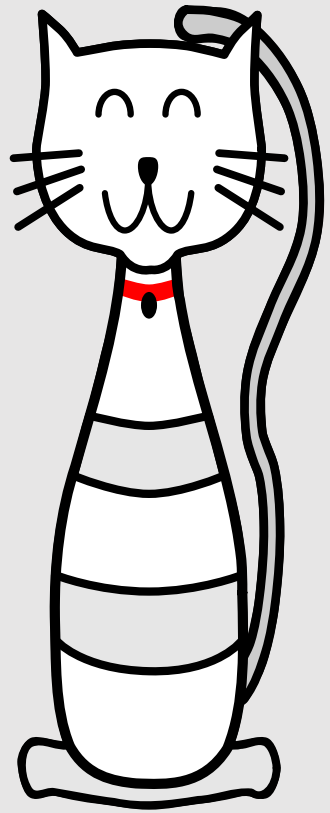
Cheshire Cat idiom in C++

- The idiom is also well known as
  - Compiler firewall idiom
  - d-pointer
  - handle classes
  - **Pimpl (pointer to implementation) idiom**



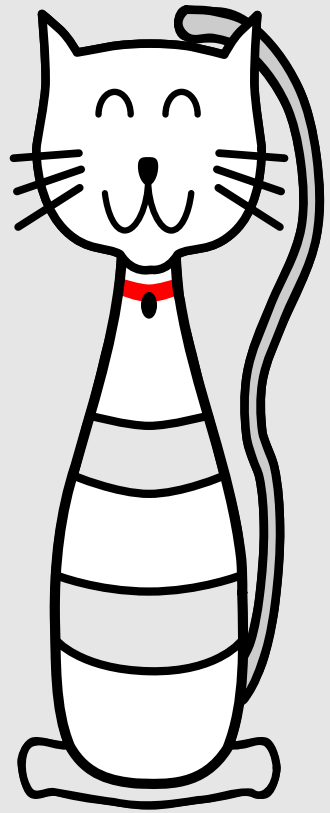
Concepts behind Pimpl Idiom

- Pimpl idiom is based on the concept of **Opaque Pointer**



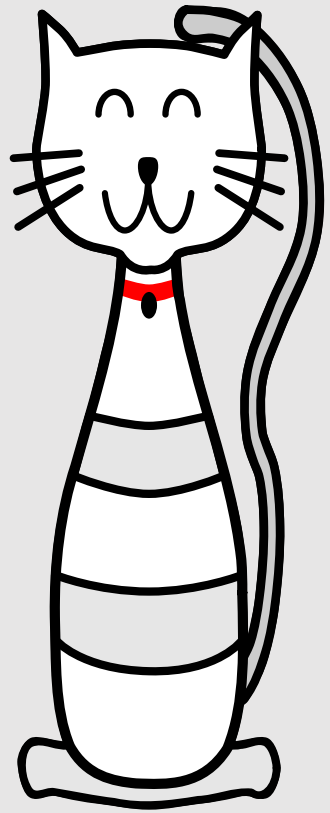
Concepts behind Pimpl Idiom

- Pimpl idiom is based on the concept of **Opaque Pointer**
- Opaque means something that can’t be seen through



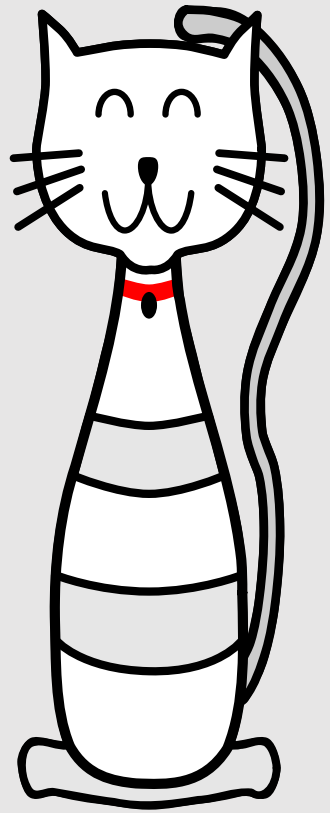
Concepts behind Pimpl Idiom

- Pimpl idiom is based on the concept of **Opaque Pointer**
- Opaque means something that can’t be seen through
- Opaque pointer is a pointer,



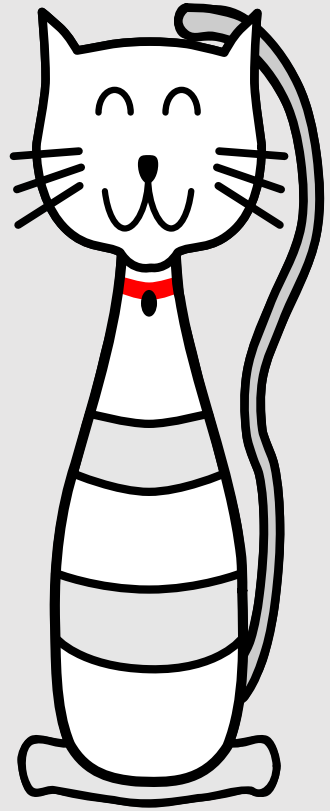
Concepts behind Pimpl Idiom

- Pimpl idiom is based on the concept of **Opaque Pointer**
- Opaque means something that can’t be seen through
- Opaque pointer is a pointer,
  - Points to a data structure



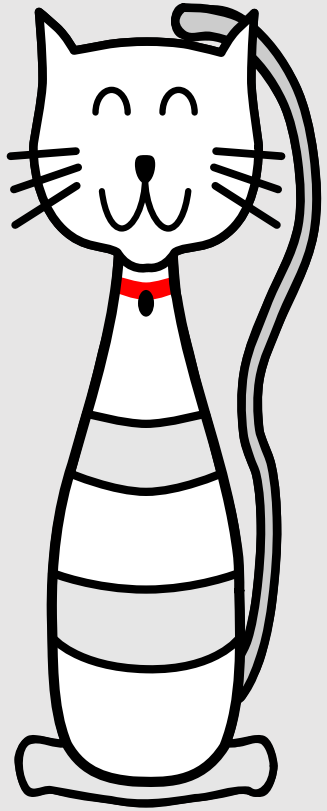
Concepts behind Pimpl Idiom

- Pimpl idiom is based on the concept of **Opaque Pointer**
- Opaque means something that can’t be seen through
- Opaque pointer is a pointer,
  - Points to a data structure
  - Contents of data struture is not exposed at the time of its definition



Concepts behind Pimpl Idiom

- Pimpl idiom is based on the concept of **Opaque Pointer**
- Opaque means something that can’t be seen through
- Opaque pointer is a pointer,
  - Points to a data structure
  - Contents of data struture is not exposed at the time of its definition
  - Hides implementation details of an interface from clients.



Examples

Aim : Wirte a program to develop a Car Application.

Example

Aim : Wirte a program to develop a Car Application.

Condition:

- All modules has to be developed as independent libraries
  - Car Frame (Frame.dll)
  - Engine (Engine.dll)
  - Gearbox (Gearbox.dll)
  - Battery (Battery.dll) etc are seperate libraries.
- Re-use these libraries to write the Car Appication.

Example

Interface without Pimpl Idiom

Car

Imports
Engine.dll
Freme.dll
Battery.dll

....

#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
...
...

Example

Interface without Pimpl Idiom

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

```
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
...
...
```

GearBox.dll

GearBox.h

```
 #include <GearAssemply.h>
class GearBox {
public:
   ChangeGear( int nGear );
   Neutral()
   Reverse();

private:
   GearAssemply m_Gears;
}
```

Example

Interface without Pimpl Idiom

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

GearBox.h

```
#include <GearAssemply.h>
class GearBox {
public:
   ChangeGear( int nGear );
   Neutral()
   Reverse();

private:
   GearAssemply m_Gears;
}
```

Example

Interface without Pimpl Idiom

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

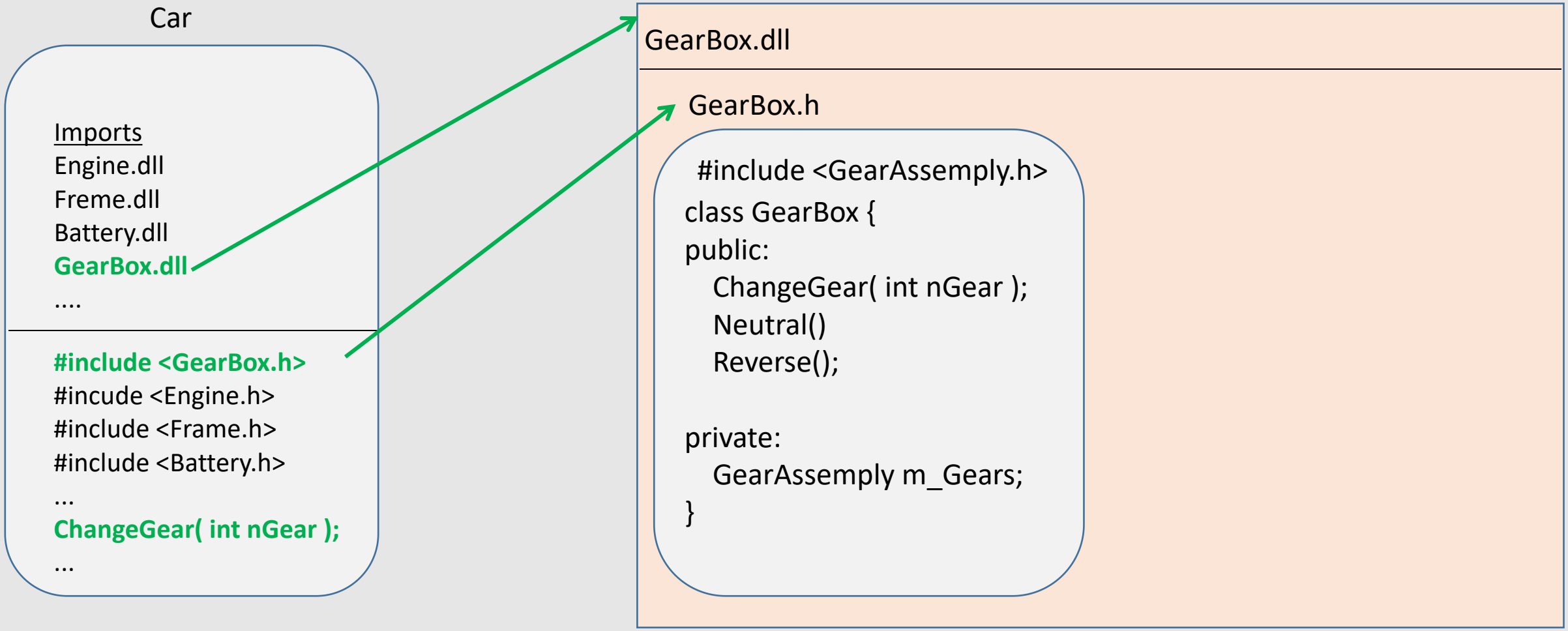
GearBox.h

```
#include <GearAssemply.h>
class GearBox {
public:
   ChangeGear( int nGear );
   Neutral()
   Reverse();

private:
   GearAssemply m_Gears;
}
```

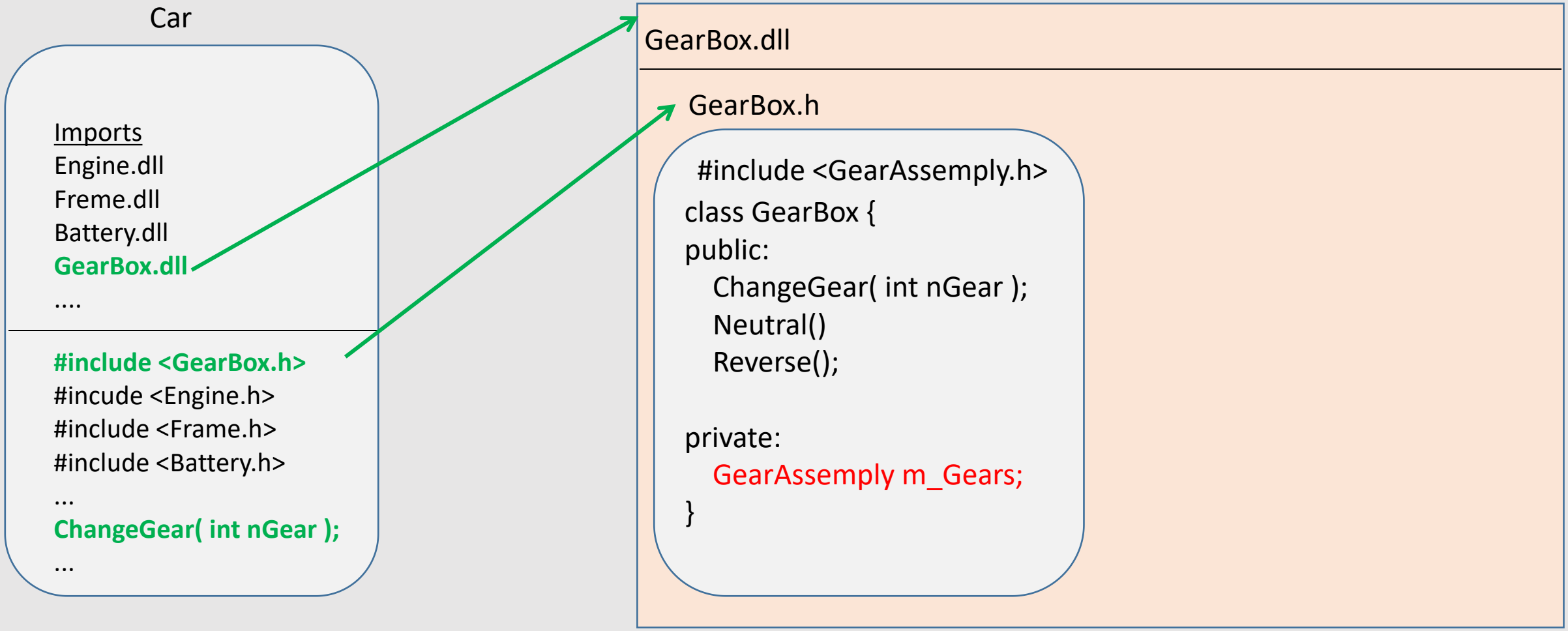
Example

Interface without Pimpl Idiom



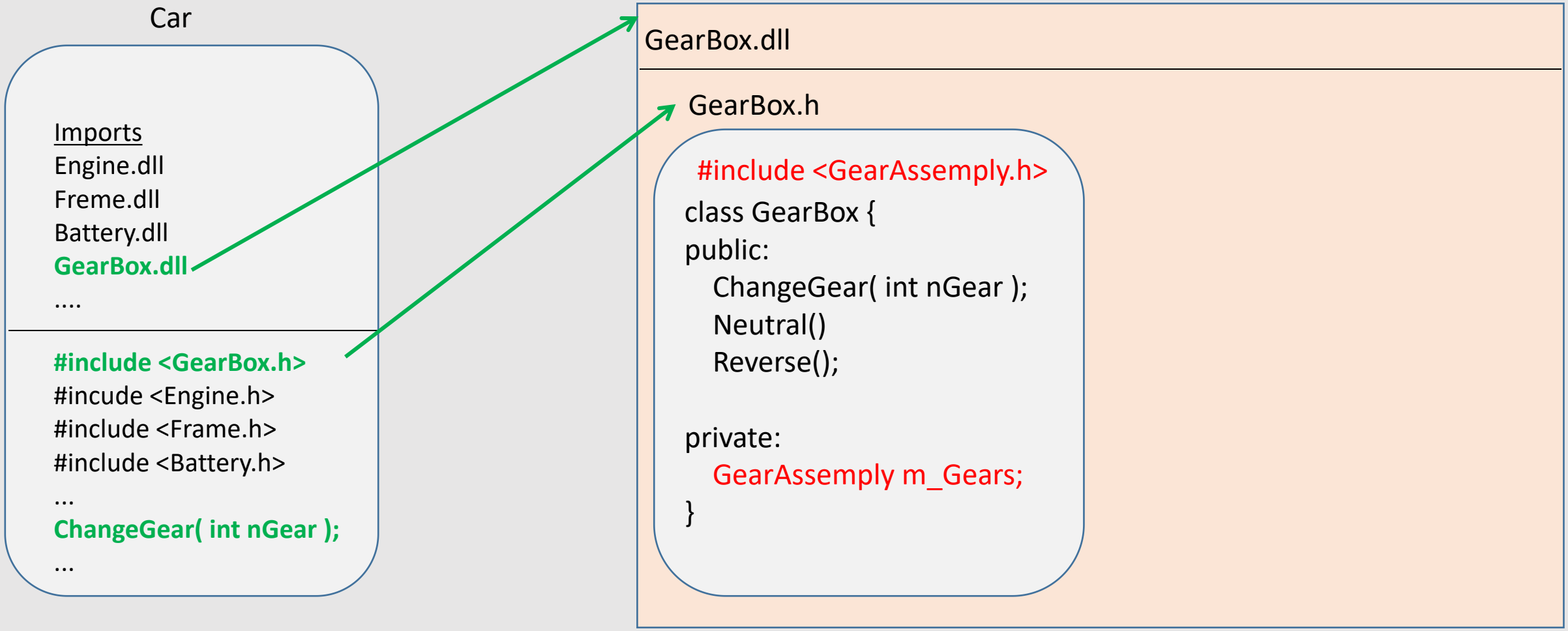
Example

Interface without Pimpl Idiom



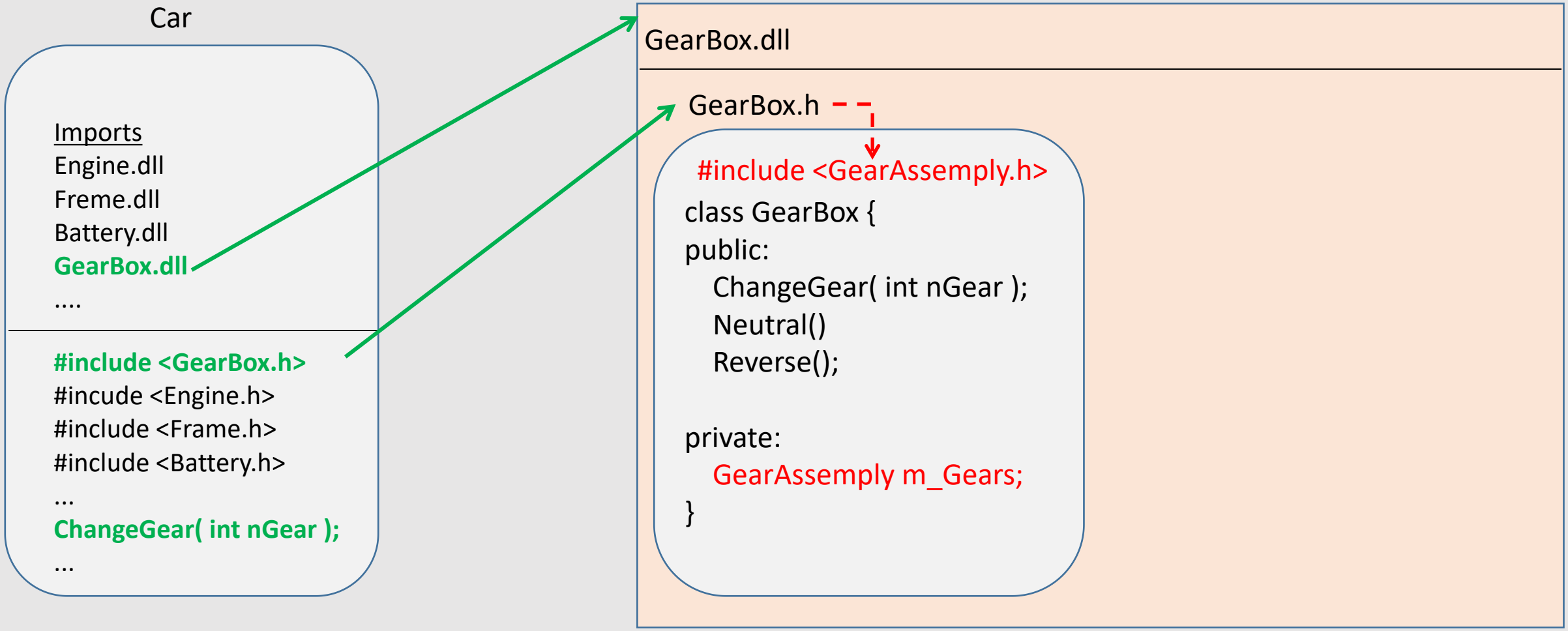
Example

Interface without Pimpl Idiom



Example

Interface without Pimpl Idiom



Example

Interface without Pimpl Idiom

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

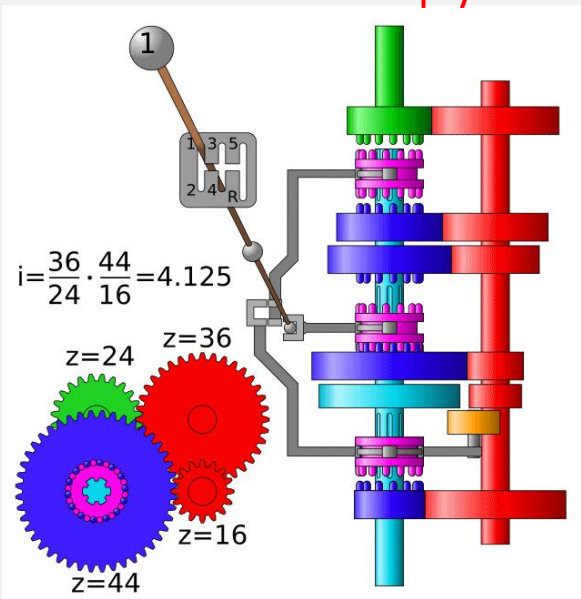
GearBox.h

```
#include <GearAssemply.h>
class GearBox {
public:
   ChangeGear( int nGear );
   Neutral()
   Reverse();

private:
   GearAssemply m_Gears;
}
```

GearAssemply.h

class GearAssemply



Example

Interface without Pimpl Idiom

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

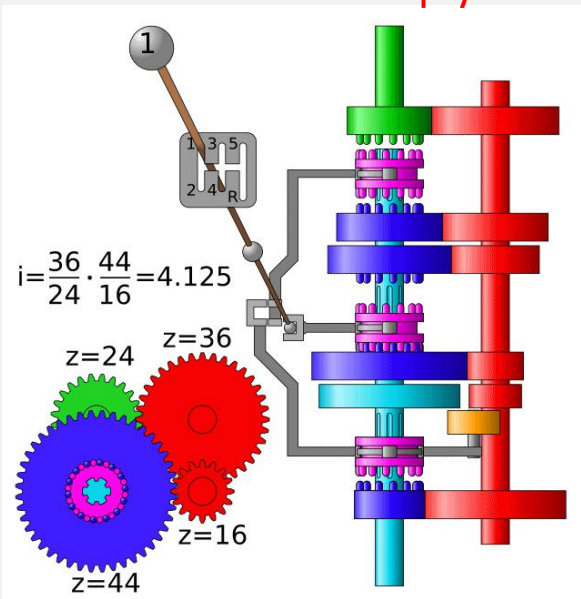
GearBox.h

```
#include <GearAssemply.h>
class GearBox {
public:
   ChangeGear( int nGear );
   Neutral()
   Reverse();

private:
   GearAssemply m_Gears;
}
```

GearAssemply.h

class GearAssemply



Example Interface without Pimpl Idiom

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

GearBox.h

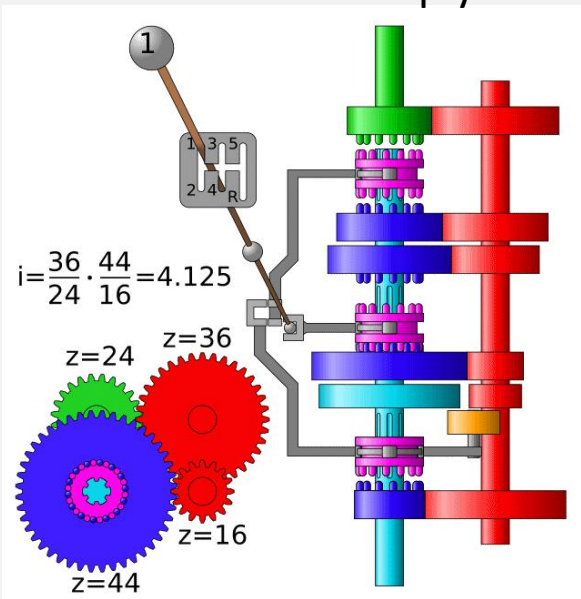
#include <GearAssemply.h>

```
class GearBox {
public:
   ChangeGear( int nGear );
   Neutral()
   Reverse();

private:
   GearAssemply m_Gears;
}
```

GearAssemply.h

class GearAssemply



Car is Tightly coupled unnecessarly with GearAssemply.h
This **direct dependency** is induced via GearBox interface

Demerits of Interface without Pimpl Idiom

Demerits of Interface without Pimpl Idiom

1. **Induced unnecessary dependency in client modules/applications**

Demerits of Interface without Pimpl Idiom

1. Induced unnecessary dependency in client modules/applications
2. **Re-build required for all Client modules/applications**

Demerits of Interface without Pimpl Idiom

1. Induced unnecessary dependency in client modules/applications
2. Re-build required for all Client modules/applications
3. **Increased number of Patch binaries**

Demerits of Interface without Pimpl Idiom

1. Induced unnecessary dependency in client modules/applications
2. Re-build required for all Client modules/applications
3. Increased number of Patch binaries
4. **Build time increased**

Demerits of Interface without Pimpl Idiom

1. Induced unnecessary dependency in client modules/applications
2. Re-build required for all Client modules/applications
3. Increased number of Patch binaries
4. Build time increased
5. **Client Impact increased**

Demerits of Interface without Pimpl Idiom

1. Induced unnecessary dependency in client modules/applications
2. Re-build required for all Client modules/applications
3. Increased number of Patch binaries
4. Build time increased
5. Client Impact increased
6. **Binary code compatibility issues increased**

Implement Pimpl Idiom in Interface

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

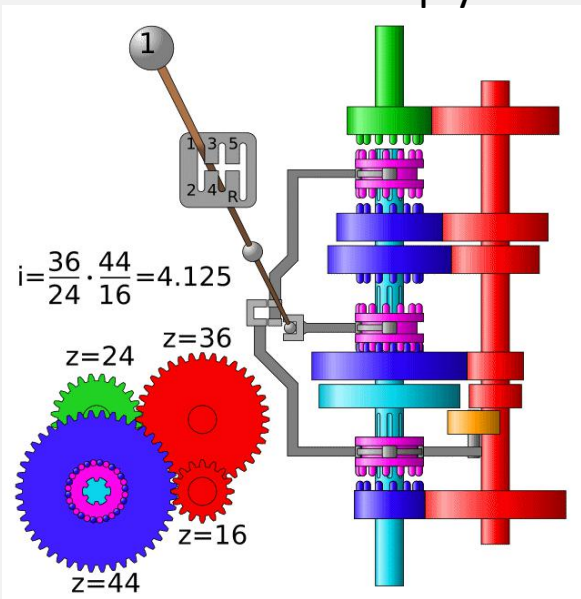
GearBox.h

```
#include <GearAssemply.h>
class GearBox {
public:
   ChangeGear( int nGear );
   Neutral()
   Reverse();

private:
   GearAssemply m_Gears;
}
```

GearAssemply.h

class GearAssemply



Car is Tightly coupled with GearAssemply module
This **direct dependency** is induced via GearBox interface

Implement Pimpl Idiom in Interface

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

GearBox.h

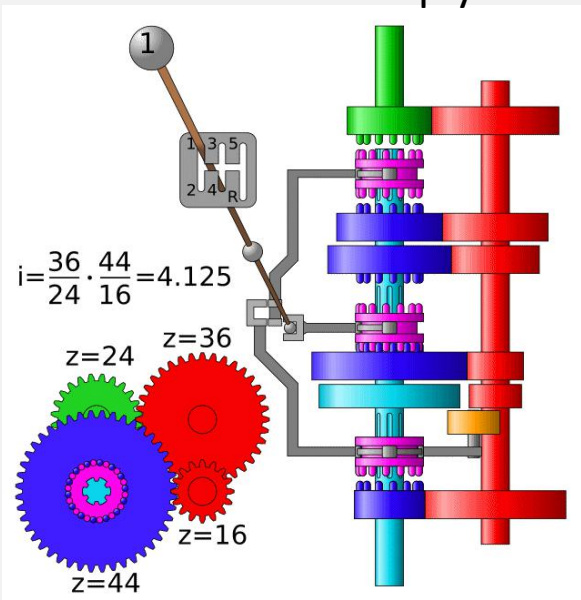
~~#include <GearAssemply.h>~~

```
class GearBox {
public:
    ChangeGear( int nGear );
    Neutral()
    Reverse();

private:
    GearAssemply m_Gears;
}
```

GearAssemply.h

class GearAssemply



Implement Pimpl Idiom in Interface

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear(int nGear);
...

GearBox.dll

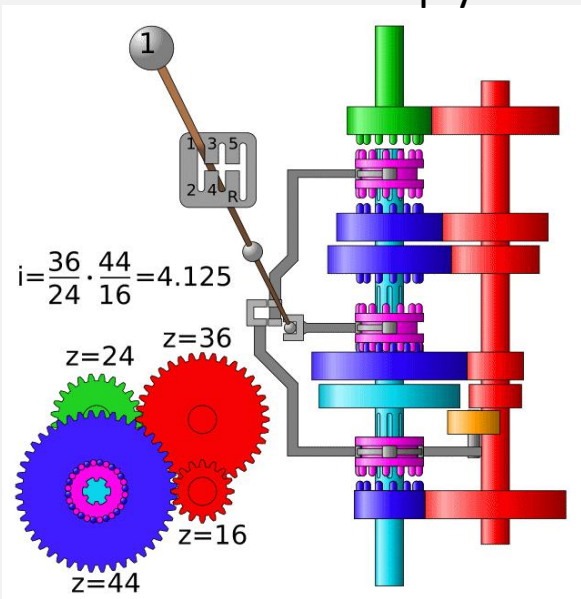
GearBox.h

~~#include <GearAssemply.h>~~
class GearBox {
public:
ChangeGear(int nGear);
Neutral()
Reverse();

private:
~~GearAssemply m_Gears;~~
class GearAssemply;
GearAssemply* m_pGearImpl
}

GearAssemply.h

class GearAssemply



Implement Pimpl Idiom in Interface

Car

Imports
Engine.dll
Freme.dll
Battery.dll
GearBox.dll
....

```
#include <GearBox.h>
#incude <Engine.h>
#include <Frame.h>
#include <Battery.h>
...
ChangeGear( int nGear );
...
```

GearBox.dll

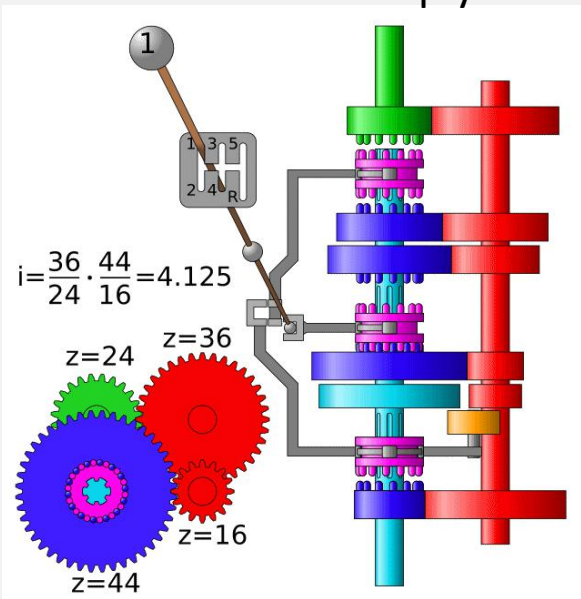
GearBox.h

```
class GearBox {
public:
    ChangeGear( int nGear );
    Neutral()
    Reverse();

private:
   class GearAssemply;
   GearAssemply* m_pGearImpl;
}
```

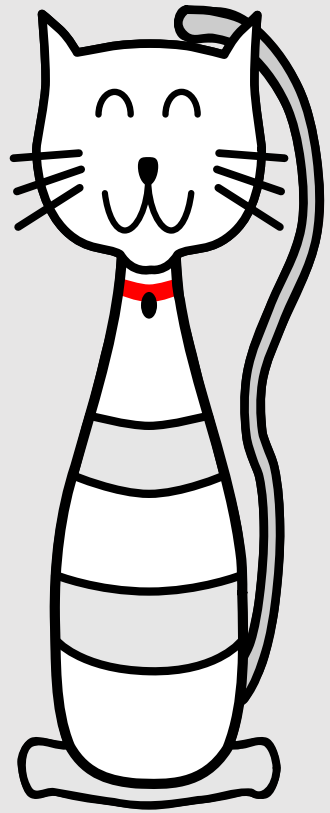
GearAssemply.h

class GearAssemply



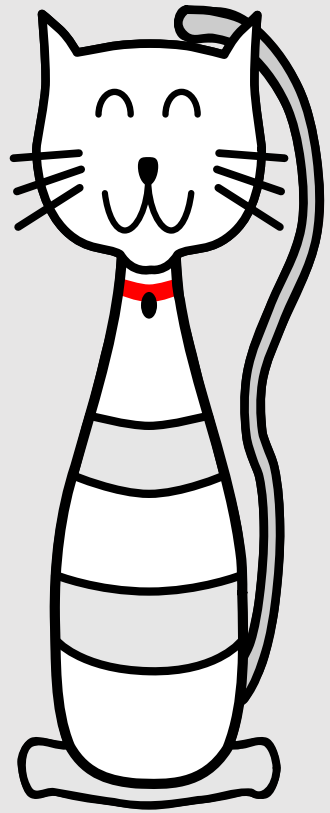
Car is **not dependent** on Gear Assemply.

Benefits of Pimpl idiom



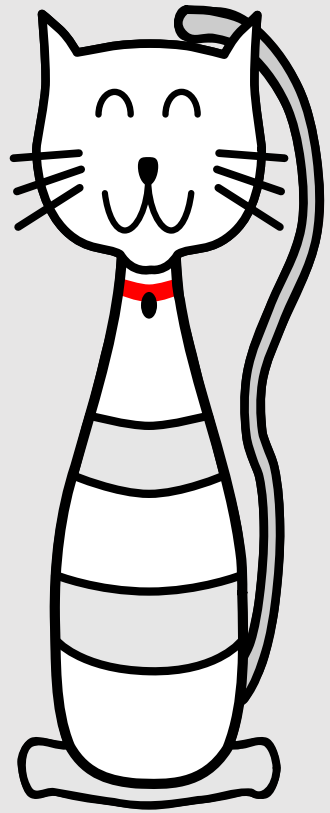
Benefits of Pimpl idiom

1. **Simpler Interface**



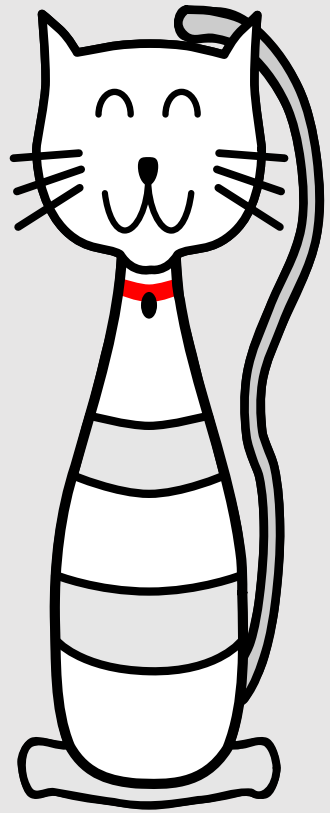
Benefits of Pimpl idiom

1. Simpler Interface
2. **Clients cannot de-reference the Opaque pointers.**



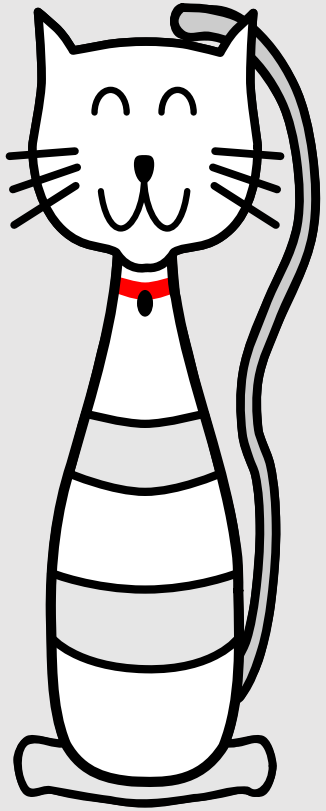
Benefits of Pimpl idiom

1. Simpler Interface
2. Clients cannot de-reference the Opaque pointers.
3. **Minimized coupling & breaks compile-time dependencies**



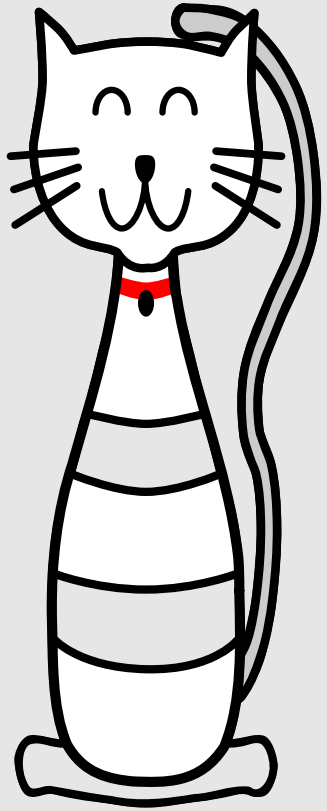
Benefits of Pimpl idiom

1. Simpler Interface
2. Clients cannot de-reference the Opaque pointers.
3. Minimized coupling & breaks compile-time dependencies
4. **Allows binary code compatibility through different versions of a shared library.**



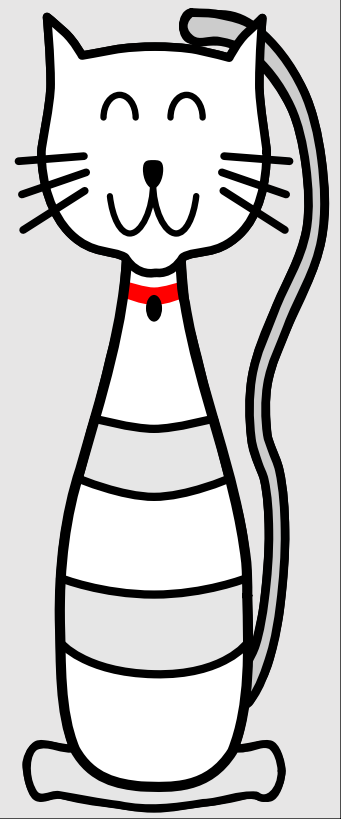
Benefits of Pimpl idiom

1. Simpler Interface
2. Clients cannot de-reference the Opaque pointers.
3. Minimized coupling & breaks compile-time dependencies
4. Allows binary code compatibility through different versions of a shared library.
5. **Reduced build time**



Benefits of Pimpl idiom

1. Simpler Interface
2. Clients cannot de-reference the Opaque pointers.
3. Minimized coupling & breaks compile-time dependencies
4. Allows binary code compatibility through different versions of a shared library.
5. Reduced build time
6. **Localized the code modification impact to the module it is not opaque.**



Demerits of Pimpl idiom

1. Increased interface object size

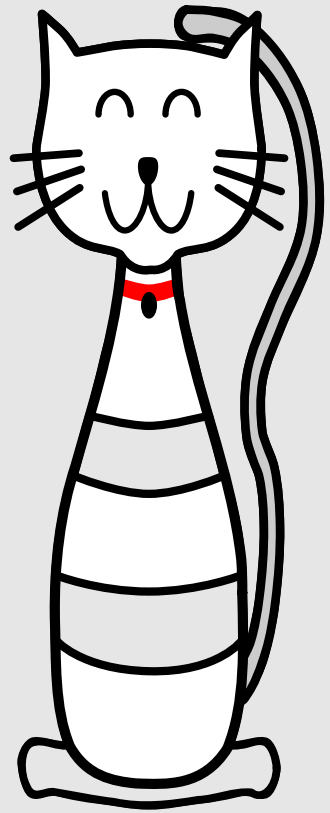
Pimpl idiom is based on usage of Opaque pointer.

The usage of Opaque Pointer causes the size of the interface increased by the size of a pointer.

4 or 8 bytes based on x86 or x64 architecture respectively.

2. Performance penality

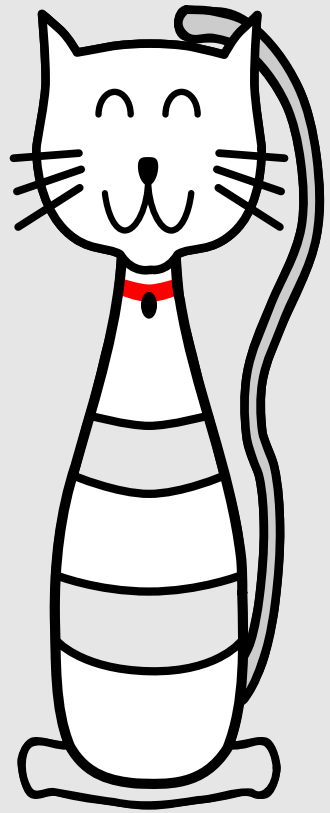
- Performance degradation can happen due to an addition memory allocation of pointer member varaible for the forward declared implementation class.
- The amount of degradation depends on the allocated objects internals



Good practices when using Pimpl idiom

1. **Use smart pointers as the Opaque Pointer.**
    1. The freeing of Opaque Pointer is automated when smart pointer is used.
    2. Use std::unique_ptr or std::shared_ptr.
    3. **Always prefer std::unique_ptr** over std::shared_ptr where ever possible.

    Mark : 3/5



Good practices when using Pimpl idiom

2. **Use an internal implementation class as Opaque pointer.**
    1. **<u>Forward delare the Implementation class type as private</u>.**

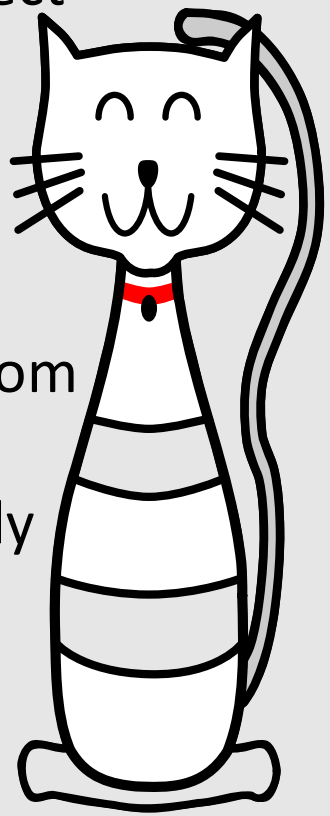
       This prevent the object creation of implementation class from other classes. I.e. only the intreface class implementation can create the object of its privatly declared class.

    2. Mirror all exposed interfaces inside the internal implementation class.

    3. Move all private members functions and private member varaibles from the exposed interface to the implementation class.

       I.e., The Exposed interface should contain only public sections with only one private Opaque pointer as the member variable.

Mark : 6/6



Client Application code

Clients shall include header file “GearBox.h” in there code.

```
1  #include "GearBox.h"
2
3  GearBox g_gearBox;
4
5  void Car( const int nOperation_i, const int nGear_i = 1 ){
6      switch( nOperation_i ){
7          case 1: { // Change Gear
8              g_gearBox.ChangeGear( nGear_i ); return;
9          }
10         case 2: { // set to Neutral
11             g_gearBox.Neutral(); return;
12         }
13         case 3: { // set to reverse
14             g_gearBox.Reverse(); return;
15         }
16         default: { // Error
17             return;
18         }
19     }
20 }
21
22 void DriveCar() {
23     Car( 1, 1 );// Change Gear to 1st gear
24     Car( 1, 2 );// Change Gear to 2nd gear
25     Car( 1, 1 );// Change Gear to 1st gear
26     Car( 2 ); // set to Neutral
27     Car( 3 ); // set to reverse
28 }
29
30 int main(){
31     DriveCar();
32     return 0;
33 }
```

GearBox.h

```
1  // Exposed Interface used by client modules
2  #pragma once
3  #include <memory>
4
5  // Interface class 'GearBox'
6  class GearBox
7  {
8  public:
9      void ChangeGear( const int nGear_i );
10     void Neutral();
11     void Reverse();
12
13     GearBox();
14     ~GearBox();
15
16 private:
17     // Forward declaration to implementation class
18     class GearBoxImpl;
19
20     // Opaque Pointer using the
21     // forward declared class type
22     std::unique_ptr<GearBoxImpl>m_GearsImpl;
23 };
```

Note:

This “GearBox.h” does not have any unwanted internal include files

GearBox.h

```
1   // Exposed Interface used by client modules
2   #pragma once
3   #include <memory>
4
5   // Interface class 'GearBox'
6   class GearBox
7   {
8   public:
9       void ChangeGear( const int nGear_i );
10      void Neutral();
11      void Reverse();
12
13      GearBox();
14      ~GearBox();
15
16  private:
17      // Forward declaration to implementation class
18      class GearBoxImpl;
19
20      // Opaque Pointer using the
21      // forward declared class type
22      std::unique_ptr<GearBoxImpl>m_GearsImpl;
23  };
```

GearBoxImpl.h

```
1   #pragma once
2   #include "GearBox.h"
3   // Internal class which should not be exposed. Just for
4   // maintainability this class is spereated to new .h and
5   // .cpp file. It is just an extension of GearBox.cpp but
6   // required for 'GearBox' interface functions.
7   #include "GearAssemply.h"
8
9   class GearBox::GearBoxImpl
10  {
11  public:
12      // Mirrored the 'GearBox' interfaces public functions
13      // inside GearBoxImpl class
14      void ChangeGear( const int nGear_i );
15      void Neutral();
16      void Reverse();
17
18      GearBoxImpl();
19      ~GearBoxImpl();
20
21  private:
22      // All the private members functions of interface
23      // 'GearBox' is moved to 'GearBoxImpl' class as private
24      int CalculateGearRatio();
25      int ValidateGearAssemply();
26
27  private:
28      // All the private members variables of interface
29      // 'GearBox'is moved to 'GearBoxImpl' class as private
30      GearAssemply m_Gears;
31      int m_nCurrentGear = 1;
32      double m_nGearRatio = 0.542;
33      int m_nGearHandlePosition = 1;
34  };
```

GearBox.h

```
1   // Exposed Interface used by client modules
2   #pragma once
3   #include <memory>
4
5   // Interface class 'GearBox'
6   class GearBox
7   {
8   public:
9       void ChangeGear( const int nGear_i );
10      void Neutral();
11      void Reverse();
12
13      GearBox();
14      ~GearBox();
15
16  private:
17      // Forward declaration to implementation class
18      class GearBoxImpl;
19
20      // Opaque Pointer using the
21      // forward declared class type
22      std::unique_ptr<GearBoxImpl>m_GearsImpl;
23  };
```

Private class forward declaration

GearBoxImpl.h

```
1   #pragma once
2   #include "GearBox.h"
3   // Internal class which should not be exposed. Just for
4   // maintainability this class is spereated to new .h and
5   // .cpp file. It is just an extension of GearBox.cpp but
6   // required for 'GearBox' interface functions.
7   #include "GearAssemply.h"
8
9   class GearBox::GearBoxImpl
10  {
11  public:
12      // Mirrored the 'GearBox' interfaces public functions
13      // inside GearBoxImpl class
14      void ChangeGear( const int nGear_i );
15      void Neutral();
16      void Reverse();
17
18      GearBoxImpl();
19      ~GearBoxImpl();
20
21  private:
22      // All the private members functions of interface
23      // 'GearBox' is moved to 'GearBoxImpl' class as private
24      int CalculateGearRatio();
25      int ValidateGearAssemply();
26
27  private:
28      // All the private members variables of interface
29      // 'GearBox'is moved to 'GearBoxImpl' class as private
30      GearAssemply m_Gears;
31      int m_nCurrentGear = 1;
32      double m_nGearRatio = 0.542;
33      int m_nGearHandlePosition = 1;
34  };
```

GearBox.h

```
1   // Exposed Interface used by client modules
2   #pragma once
3   #include <memory>
4
5   // Interface class 'GearBox'
6   class GearBox
7   {
8   public:
9       void ChangeGear( const int nGear_i );
10      void Neutral();
11      void Reverse();
12
13      GearBox();
14      ~GearBox();
15
16  private:
17      // Forward declaration to implementation class
18      class GearBoxImpl;
19
20      // Opaque Pointer using the
21      // forward declared class type
22      std::unique_ptr<GearBoxImpl>m_GearsImpl;
23  };
```

Opaque Pointer using std::unique_ptr

GearBoxImpl.h

```
1   #pragma once
2   #include "GearBox.h"
3   // Internal class which should not be exposed. Just for
4   // maintainability this class is spereated to new .h and
5   // .cpp file. It is just an extension of GearBox.cpp but
6   // required for 'GearBox' interface functions.
7   #include "GearAssemply.h"
8
9   class GearBox::GearBoxImpl
10  {
11  public:
12      // Mirrored the 'GearBox' interfaces public functions
13      // inside GearBoxImpl class
14      void ChangeGear( const int nGear_i );
15      void Neutral();
16      void Reverse();
17
18      GearBoxImpl();
19      ~GearBoxImpl();
20
21  private:
22      // All the private members functions of interface
23      // 'GearBox' is moved to 'GearBoxImpl' class as private
24      int CalculateGearRatio();
25      int ValidateGearAssemply();
26
27  private:
28      // All the private members variables of interface
29      // 'GearBox'is moved to 'GearBoxImpl' class as private
30      GearAssemply m_Gears;
31      int m_nCurrentGear = 1;
32      double m_nGearRatio = 0.542;
33      int m_nGearHandlePosition = 1;
34  };
```

GearBox.h

```
1   // Exposed Interface used by client modules
2   #pragma once
3   #include <memory>
4
5   // Interface class 'GearBox'
6   class GearBox
7   {
8   public:
9       void ChangeGear( const int nGear_i );
10      void Neutral();
11      void Reverse();
12
13      GearBox();
14      ~GearBox();
15
16  private:
17      // Forward declaration to implementation class
18      class GearBoxImpl;
19
20      // Opaque Pointer using the
21      // forward declared class type
22      std::unique_ptr<GearBoxImpl>m_GearsImpl;
23  };
```

GearBoxImpl.h

```
1   #pragma once
2   #include "GearBox.h"
3   // Internal class which should not be exposed. Just for
4   // maintainability this class is spereated to new .h and
5   // .cpp file. It is just an extension of GearBox.cpp but
6   // required for 'GearBox' interface functions.
7   #include "GearAssemply.h"
8
9   class GearBox::GearBoxImpl
10  {
11  public:
12      // Mirrored the 'GearBox' interfaces public functions
13      // inside GearBoxImpl class
14      void ChangeGear( const int nGear_i );
15      void Neutral();
16      void Reverse();
17
18      GearBoxImpl();
19      ~GearBoxImpl();
20
21  private:
22      // All the private members functions of interface
23      // 'GearBox' is moved to 'GearBoxImpl' class as private
24      int CalculateGearRatio();
25      int ValidateGearAssemply();
26
27  private:
28      // All the private members variables of interface
29      // 'GearBox'is moved to 'GearBoxImpl' class as private
30      GearAssemply m_Gears;
31      int m_nCurrentGear = 1;
32      double m_nGearRatio = 0.542;
33      int m_nGearHandlePosition = 1;
34  };
```

Private class

Note: Its object can be created only inside GearBox.cpp

GearBox.h

```
1   // Exposed Interface used by client modules
2   #pragma once
3   #include <memory>
4
5   // Interface class 'GearBox'
6   class GearBox
7   {
8   public:
9       void ChangeGear( const int nGear_i );
10      void Neutral();
11      void Reverse();
12
13      GearBox();
14      ~GearBox();
15
16  private:
17      // Forward declaration to implementation class
18      class GearBoxImpl;
19
20      // Opaque Pointer using the
21      // forward declared class type
22      std::unique_ptr<GearBoxImpl>m_GearsImpl;
23  };
```

GearBoxImpl.h

```
1   #pragma once
2   #include "GearBox.h"
3   // Internal class which should not be exposed. Just for
4   // maintainability this class is spereated to new .h and
5   // .cpp file. It is just an extension of GearBox.cpp but
6   // required for 'GearBox' interface functions.
7   #include "GearAssemply.h"
8
9   class GearBox::GearBoxImpl
10  {
11  public:
12      // Mirrored the 'GearBox' interfaces public functions
13      // inside GearBoxImpl class
14      void ChangeGear( const int nGear_i );
15      void Neutral();
16      void Reverse();
17
18      GearBoxImpl();
19      ~GearBoxImpl();
20
21  private:
22      // All the private members functions of interface
23      // 'GearBox' is moved to 'GearBoxImpl' class as private
24      int CalculateGearRatio();
25      int ValidateGearAssemply();
26
27  private:
28      // All the private members variables of interface
29      // 'GearBox'is moved to 'GearBoxImpl' class as private
30      GearAssemply m_Gears;
31      int m_nCurrentGear = 1;
32      double m_nGearRatio = 0.542;
33      int m_nGearHandlePosition = 1;
34  };
```

GearBox.cpp

```cpp
// Important:
// First Include the header file in which
// GearBoxImpl class is declared.
#include "GearBoxImpl.h"

// Interface header file.
#include "GearBox.h"

GearBox::GearBox(){
    m_GearsImpl = std::make_unique<GearBoxImpl>();
}

GearBox::~GearBox() = default;

void GearBox::ChangeGear(const int nGear_i){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::ChangeGear() function
    m_GearsImpl->ChangeGear( nGear_i );
}

void GearBox::Neutral(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Neutral() function
    m_GearsImpl->Neutral();
}

void GearBox::Reverse(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Reverse() function
    m_GearsImpl->Reverse();
}
```

GearBoxImpl.cpp

```cpp
#include "GearBoxImpl.h"
#include <iostream>
void GearBox::GearBoxImpl::ChangeGear( const int nGear_i ){
    // Actual implementation of ChangeGear() functionality
    std::cout << "ChangeGear(const int nGear_i)\n";
}

void GearBox::GearBoxImpl::Neutral(){
    // Actual implementation of Neutral() functionality
    std::cout << "Neutral()\n";
}

void GearBox::GearBoxImpl::Reverse(){
    // Actual implementation of Reverse() functionality
    std::cout << "Reverse()\n";
}

GearBox::GearBoxImpl::GearBoxImpl()
{
    std::cout << "GearBoxImpl Ctor\n";
}

GearBox::GearBoxImpl::~GearBoxImpl()
{
        std::cout << "GearBoxImpl Dtor\n";
}
```

GearBox.cpp

```cpp
// Important:
// First Include the header file in which
// GearBoxImpl class is declared.
#include "GearBoxImpl.h"

// Interface header file.
#include "GearBox.h"

GearBox::GearBox(){
    m_GearsImpl = std::make_unique<GearBoxImpl>();
}

GearBox::~GearBox() = default;

void GearBox::ChangeGear(const int nGear_i){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::ChangeGear() function
    m_GearsImpl->ChangeGear( nGear_i );
}

void GearBox::Neutral(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Neutral() function
    m_GearsImpl->Neutral();
}

void GearBox::Reverse(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Reverse() function
    m_GearsImpl->Reverse();
}
```

GearBoxImpl.cpp

```cpp
#include "GearBoxImpl.h"
#include <iostream>
void GearBox::GearBoxImpl::ChangeGear( const int nGear_i ){
    // Actual implementation of ChangeGear() functionality
    std::cout << "ChangeGear(const int nGear_i)\n";
}

void GearBox::GearBoxImpl::Neutral(){
    // Actual implementation of Neutral() functionality
    std::cout << "Neutral()\n";
}

void GearBox::GearBoxImpl::Reverse(){
    // Actual implementation of Reverse() functionality
    std::cout << "Reverse()\n";
}

GearBox::GearBoxImpl::GearBoxImpl()
{
    std::cout << "GearBoxImpl Ctor\n";
}

GearBox::GearBoxImpl::~GearBoxImpl()
{
        std::cout << "GearBoxImpl Dtor\n";
}
```

GearBox.cpp

```cpp
// Important:
// First Include the header file in which
// GearBoxImpl class is declared.
#include "GearBoxImpl.h"

// Interface header file.
#include "GearBox.h"

GearBox::GearBox(){
    m_GearsImpl = std::make_unique<GearBoxImpl>();
}

GearBox::~GearBox() = default;

void GearBox::ChangeGear(const int nGear_i){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::ChangeGear() function
    m_GearsImpl->ChangeGear( nGear_i );
}

void GearBox::Neutral(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Neutral() function
    m_GearsImpl->Neutral();
}

void GearBox::Reverse(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Reverse() function
    m_GearsImpl->Reverse();
}
```

GearBoxImpl.cpp

```cpp
#include "GearBoxImpl.h"
#include <iostream>
void GearBox::GearBoxImpl::ChangeGear( const int nGear_i ){
    // Actual implementation of ChangeGear() functionality
    std::cout << "ChangeGear(const int nGear_i)\n";
}

void GearBox::GearBoxImpl::Neutral(){
    // Actual implementation of Neutral() functionality
    std::cout << "Neutral()\n";
}

void GearBox::GearBoxImpl::Reverse(){
    // Actual implementation of Reverse() functionality
    std::cout << "Reverse()\n";
}

GearBox::GearBoxImpl::GearBoxImpl()
{
    std::cout << "GearBoxImpl Ctor\n";
}

GearBox::GearBoxImpl::~GearBoxImpl()
{
        std::cout << "GearBoxImpl Dtor\n";
}
```

GearBox.cpp

```cpp
// Important:
// First Include the header file in which
// GearBoxImpl class is declared.
#include "GearBoxImpl.h"

// Interface header file.
#include "GearBox.h"

GearBox::GearBox(){
    m_GearsImpl = std::make_unique<GearBoxImpl>();
}

GearBox::~GearBox() = default;

void GearBox::ChangeGear(const int nGear_i){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::ChangeGear() function
    m_GearsImpl->ChangeGear( nGear_i );
}

void GearBox::Neutral(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Neutral() function
    m_GearsImpl->Neutral();
}

void GearBox::Reverse(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Reverse() function
    m_GearsImpl->Reverse();
}
```

GearBoxImpl.cpp

```cpp
#include "GearBoxImpl.h"
#include <iostream>
void GearBox::GearBoxImpl::ChangeGear( const int nGear_i ){
    // Actual implementation of ChangeGear() functionality
    std::cout << "ChangeGear(const int nGear_i)\n";
}

void GearBox::GearBoxImpl::Neutral(){
    // Actual implementation of Neutral() functionality
    std::cout << "Neutral()\n";
}

void GearBox::GearBoxImpl::Reverse(){
    // Actual implementation of Reverse() functionality
    std::cout << "Reverse()\n";
}

GearBox::GearBoxImpl::GearBoxImpl()
{
    std::cout << "GearBoxImpl Ctor\n";
}

GearBox::GearBoxImpl::~GearBoxImpl()
{
        std::cout << "GearBoxImpl Dtor\n";
}
```

GearBox.cpp

```cpp
// Important:
// First Include the header file in which
// GearBoxImpl class is declared.
#include "GearBoxImpl.h"

// Interface header file.
#include "GearBox.h"

GearBox::GearBox(){
    m_GearsImpl = std::make_unique<GearBoxImpl>();
}

GearBox::~GearBox() = default;

void GearBox::ChangeGear(const int nGear_i){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::ChangeGear() function
    m_GearsImpl->ChangeGear( nGear_i );
}

void GearBox::Neutral(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Neutral() function
    m_GearsImpl->Neutral();
}

void GearBox::Reverse(){
    // Use the m_pGearsImpl and delegate to
    // GearBoxImpl::Reverse() function
    m_GearsImpl->Reverse();
}
```

GearBoxImpl.cpp

```cpp
#include "GearBoxImpl.h"
#include <iostream>
void GearBox::GearBoxImpl::ChangeGear( const int nGear_i ){
    // Actual implementation of ChangeGear() functionality
    std::cout << "ChangeGear(const int nGear_i)\n";
}

void GearBox::GearBoxImpl::Neutral(){
    // Actual implementation of Neutral() functionality
    std::cout << "Neutral()\n";
}

void GearBox::GearBoxImpl::Reverse(){
    // Actual implementation of Reverse() functionality
    std::cout << "Reverse()\n";
}

GearBox::GearBoxImpl::GearBoxImpl()
{
    std::cout << "GearBoxImpl Ctor\n";
}

GearBox::GearBoxImpl::~GearBoxImpl()
{
        std::cout << "GearBoxImpl Dtor\n";
}
```

Thank You

LIKE & SHARE &

Subscribe to our YouTube Channel

https://youtu.be/oTpGyu2L8C8

Attribution

- https://commons.wikimedia.org/wiki/File:Gearbox_4gears.gif