RAII, PIMPL

ИДИОМЫ ПРОГРАММИРОВАНИЯ

ИДИОМА RAII — ЗАХВАТ РЕСУРСА ЕСТЬ ИНИЦИАЛИЗАЦИЯ

Идиома очень простая и кратко описывается следующим образом: в конструкторе объект получает доступ к какому либо ресурсу и сохраняет описатель ресурса в закрытый член класса, а при вызове деструктура этот ресурс освобождается. При объявлении объекта данного класса на стеке происходит и его инициализация с вызовом конструктора, захватывающий ресурс. При выходе из области видимости объект выталкивается из стека, но перед этим вызывается деструктор объекта, который и освобождает захваченный ресурс.

```
1. class TelephoneLine
2. {
3. public:
4.    void pickUpThePhoneUp()
5.    {
6.       std::cout << "Line locked\n";
7.    }
8.    void putThePhoneDown()
9.    {
10.      std::cout << "Line unlocked\n";
11.   }
12. };
```

```
1. class TelephoneCall
2. {
3. public:
4.    TelephoneCall()
5.    {
6.       telephoneLine = new TelephoneLine();
7.       telephoneLine->pickUpThePhoneUp();
8.    }
9.    ~TelephoneCall()
10.   {
11.      telephoneLine->putThePhoneDown();
12.      delete telephoneLine;
13.   }
14. private:
15.   TelephoneCall (const TelephoneCall &);
16.   TelephoneCall& operator=(const TelephoneCall &);
17.   TelephoneLine * telephoneLine;
18. };
```

```
1. int main()
2. {
3.   {
4.       std::cout << "Let's make a call to a friend.\n";
5.       TelephoneCall call;
6.       std::cout << "Oh, we've talked enough. I need to
   take a nap. Goodbye!\n";
7.   }
8.   std::cout << "Zzzzzz...";
9. }
```

После запуска мы увидим следующий вывод:

Let's make a call to a friend.
 Line locked
 Oh, we've talked enough. I need to take a nap.
Goodbye!
 Line unlocked
 Zzzzzz...

PIMPL

Допустим, нам необходимо написать кроссплатформенное сетевое приложение с использованием сокетов. Для этого нам необходим класс GeneralSocket (“Видимый класс”), который будет инкапсулировать в себе детали реализации конкретной платформы (“Скрываемый класс”). Часто требуется скрыть детали реализации от пользователей или других разработчиков:

- Для того, что бы была возможность изменять реализацию скрываемого класса без перекомпиляции остального кода, так как закрытые члены хоть и недоступны извне никому, кроме функций-членов и друзей, но видимы всем, кто имеет доступ к определению класса. Изменение определения класса приводит к необходимости перекомпиляции всех пользователей класса
- Для сокрытия имен из области видимости. Закрытые члены хоть и не могут быть вызваны кодом вне класса, тем не менее они участвуют в поиске имен и разрешении перегрузок
- Для ускорения времени сборки, так как компилятору не нужно обрабатывать лишние определения закрытых типов

```
//GeneralSocket.h
 #include “UnixSocketImpl.h”

Class GeneralSocket{
 public:
  connect();
 private:
  UnixSocketImpl socket;
 }

//GeneralSocket.cxx
 GeneralSocket::connect(){
  socket.connectImpl();
 }
```

```
//GeneralSocket.h
 Class UnixSocketImpl;

Class GeneralSocket
 {
 public:
  GeneralSocket();
  void connect();
 private:
  UnixSocketImpl * socket;
 }
```

```
//GeneralSocket.cxx
 #include “UnixSocketImpl.h”

GeneralSocket::GeneralSocket() :
socket (new UnixSocketImpl){}

GeneralSocket::~GeneralSocket() {
  delete socket;
  socket = 0;
 }

void GeneralSocket::connect() {
  socket->connectImpl();
 }
```

НЕДОСТАТКИ:

1. Каждое создание объекта требует динамического выделения памяти для объекта, на который ссылается указатель
2. Использование нескольких уровней косвенности (как минимум — один) для доступа к членам скрытого объекта