```
 1
 2  #include<windows.h>
 3  #include"ContainmentInnerComponentWithRegFile.h"
 4  // class declarations
 5  class CMultiplicationDivision:public IMultiplication,IDivision
 6  {
 7  private:
 8      long m_cRef;
 9  public:
10      // constructor method declarations
11      CMultiplicationDivision(void);
12      // destructor method declarations
13      ~CMultiplicationDivision(void);
14      // IUnknown specific method declarations (inherited)
15      HRESULT __stdcall QueryInterface(REFIID,void **);
16      ULONG __stdcall AddRef(void);
17      ULONG __stdcall Release(void);
18      // IMultiplication specific method declarations (inherited)
19      HRESULT __stdcall MultiplicationOfTwoIntegers(int,int,int *);
20      // IDivision specific method declarations (inherited)
21      HRESULT __stdcall DivisionOfTwoIntegers(int,int,int *);
22  };
23  class CMultiplicationDivisionClassFactory:public IClassFactory
24  {
25  private:
26      long m_cRef;
27  public:
28      // constructor method declarations
29      CMultiplicationDivisionClassFactory(void);
30      // destructor method declarations
31      ~CMultiplicationDivisionClassFactory(void);
32      // IUnknown specific method declarations (inherited)
33      HRESULT __stdcall QueryInterface(REFIID,void **);
34      ULONG __stdcall AddRef(void);
35      ULONG __stdcall Release(void);
36      // IClassFactory specific method declarations (inherited)
37      HRESULT __stdcall CreateInstance(IUnknown *,REFIID,void **);
38      HRESULT __stdcall LockServer(BOOL);
39  };
40  // global variable declarations
41  long glNumberOfActiveComponents=0;// number of active components
42  long glNumberOfServerLocks=0;// number of locks on this dll
43  // DllMain
44  BOOL WINAPI DllMain(HINSTANCE hDll,DWORD dwReason,LPVOID Reserved)
45  {
46      // code
47      switch(dwReason)
48      {
49      case DLL_PROCESS_ATTACH:
50          break;
51      case DLL_PROCESS_DETACH:
52          break;
```

```
53      }
54  return(TRUE);
55  }
56  // Implementation Of CMultiplicationDivision's Constructor Method
57  CMultiplicationDivision::CMultiplicationDivision(void)
58  {
59      // code
60      m_cRef=1;// hardcoded initialization to anticipate possible failure of
          QueryInterface()
61      InterlockedIncrement(&glNumberOfActiveComponents);// increment global counter
62  }
63  // Implementation Of CSumSubtract's Destructor Method
64  CMultiplicationDivision::~CMultiplicationDivision(void)
65  {
66      // code
67      InterlockedDecrement(&glNumberOfActiveComponents);// decrement global counter
68  }
69  // Implementation Of CMultiplicationDivision's IUnknown's Methods
70  HRESULT CMultiplicationDivision::QueryInterface(REFIID riid,void **ppv)
71  {
72      // code
73      if(riid==IID_IUnknown)
74          *ppv=static_cast<IMultiplication *>(this);
75      else if(riid==IID_IMultiplication)
76          *ppv=static_cast<IMultiplication *>(this);
77      else if(riid==IID_IDivision)
78          *ppv=static_cast<IDivision *>(this);
79      else
80      {
81          *ppv=NULL;
82          return(E_NOINTERFACE);
83      }
84      reinterpret_cast<IUnknown *>(*ppv)->AddRef();
85      return(S_OK);
86  }
87  ULONG CMultiplicationDivision::AddRef(void)
88  {
89      // code
90      InterlockedIncrement(&m_cRef);
91      return(m_cRef);
92  }
93  ULONG CMultiplicationDivision::Release(void)
94  {
95      // code
96      InterlockedDecrement(&m_cRef);
97      if(m_cRef==0)
98      {
99          delete(this);
100         return(0);
101     }
102     return(m_cRef);
103 }
```

```
104 // Implementation Of IMultiplication's Methods
105 HRESULT CMultiplicationDivision::MultiplicationOfTwoIntegers(int num1,int
      num2,int *pMultiplication)
106 {
107     // code
108     *pMultiplication=num1*num2;
109     return(S_OK);
110 }
111 // Implementation Of IDivision's Methods
112 HRESULT CMultiplicationDivision::DivisionOfTwoIntegers(int num1,int num2,int
      *pDivision)
113 {
114     // code
115     *pDivision=num1/num2;
116     return(S_OK);
117 }
118 // Implementation Of CMultiplicationDivisionClassFactory's Constructor Method
119 CMultiplicationDivisionClassFactory::CMultiplicationDivisionClassFactory(void)
120 {
121     // code
122     m_cRef=1;// hardcoded initialization to anticipate possible failure of
        QueryInterface()
123 }
124 // Implementation Of CMultiplicationDivisionClassFactory's Destructor Method
125 CMultiplicationDivisionClassFactory::~CMultiplicationDivisionClassFactory(void)
126 {
127     // code
128 }
129 // Implementation Of CMultiplicationDivisionClassFactory's IClassFactory's
      IUnknown's Methods
130 HRESULT CMultiplicationDivisionClassFactory::QueryInterface(REFIID riid,void
      **ppv)
131 {
132     // code
133     if(riid==IID_IUnknown)
134         *ppv=static_cast<IClassFactory *>(this);
135     else if(riid==IID_IClassFactory)
136         *ppv=static_cast<IClassFactory *>(this);
137     else
138     {
139         *ppv=NULL;
140         return(E_NOINTERFACE);
141     }
142     reinterpret_cast<IUnknown *>(*ppv)->AddRef();
143     return(S_OK);
144 }
145 ULONG CMultiplicationDivisionClassFactory::AddRef(void)
146 {
147     // code
148     InterlockedIncrement(&m_cRef);
149     return(m_cRef);
150 }
```

```
151 ULONG CMultiplicationDivisionClassFactory::Release(void)
152 {
153     // code
154     InterlockedDecrement(&m_cRef);
155     if(m_cRef==0)
156     {
157         delete(this);
158         return(0);
159     }
160     return(m_cRef);
161 }
162 // Implementation Of CMultiplicationDivisionClassFactory's IClassFactory's
      Methods
163 HRESULT CMultiplicationDivisionClassFactory::CreateInstance(IUnknown
      *pUnkOuter,REFIID riid,void **ppv)
164 {
165     // variable declarations
166     CMultiplicationDivision *pCMultiplicationDivision=NULL;
167     HRESULT hr;
168     // code
169     if(pUnkOuter!=NULL)
170         return(CLASS_E_NOAGGREGATION);
171     // create the instance of component i.e. of CMultiplicationDivision class
172     pCMultiplicationDivision=new CMultiplicationDivision;
173     if(pCMultiplicationDivision==NULL)
174         return(E_OUTOFMEMORY);
175     // get the requested interface
176     hr=pCMultiplicationDivision->QueryInterface(riid,ppv);
177     pCMultiplicationDivision->Release();// anticipate possible failure of
        QueryInterface()
178     return(hr);
179 }
180 HRESULT CMultiplicationDivisionClassFactory::LockServer(BOOL fLock)
181 {
182     // code
183     if(fLock)
184         InterlockedIncrement(&glNumberOfServerLocks);
185     else
186         InterlockedDecrement(&glNumberOfServerLocks);
187     return(S_OK);
188 }
189 // Implementation Of Exported Functions From This Dll
190 HRESULT __stdcall DllGetClassObject(REFCLSID rclsid,REFIID riid,void **ppv)
191 {
192     // variable declaraions
193     CMultiplicationDivisionClassFactory
        *pCMultiplicationDivisionClassFactory=NULL;
194     HRESULT hr;
195     // code
196     if(rclsid!=CLSID_MultiplicationDivision)
197         return(CLASS_E_CLASSNOTAVAILABLE);
198     // create class factory
```

```
199     pCMultiplicationDivisionClassFactory=new CMultiplicationDivisionClassFactory;
200     if(pCMultiplicationDivisionClassFactory==NULL)
201         return(E_OUTOFMEMORY);
202     hr=pCMultiplicationDivisionClassFactory->QueryInterface(riid,ppv);
203     pCMultiplicationDivisionClassFactory->Release();// anticipate possible
          failure of QueryInterface()
204     return(hr);
205 }
206 HRESULT __stdcall DllCanUnloadNow(void)
207 {
208     // code
209     if((glNumberOfActiveComponents==0) && (glNumberOfServerLocks==0))
210         return(S_OK);
211     else
212         return(S_FALSE);
213 }
214
```