Lec01.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec01
{
    class Fibo
    {
        static int fibo(int n)
        {
            if (n == 0) return 1;
            else if (n == 1) return 1;
            else if (n > 0) return fibo(n - 1) + fibo(n - 2);
            else
            {
                Console.WriteLine("Unexpected minus argument.");
                Environment.Exit(-1);
                return -1;
            }
        }
        static void Main(string[] args)
        {
            for (int i = 0; i <= 10; i++)
            {
                Console.WriteLine("F(" + i + ") = " + fibo(i));
            }
        }
    }
}
```

Lec02.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec02
{
    class Stack
    {
        // Attributes
        static int MAX = 100;
        /** an array to save stack contents
         */
        private int[] _s;
        /** the index to point top of stack
         */
        private int _top;
        /** size of the stack
         */
        private int _size;
        // Operations
        /** initialization procedure for new stack
         */
        private void initialize()
        {
            // NOTE: We don't have to do this initialization with Java
            for (int i = 0; i < _size; i++)
            {
                _s[i] = 0;
            }
        }
        /** this function is called for stack overflow exception
         */
        private void overflowError()
        {
            Console.WriteLine("Stack overflow error occurs.");
            Environment.Exit(-1);
        }
        /** this function is called for stack empty exception
         */
        private void emptyError()
        {
            Console.WriteLine("Stack empty error occurs.");
            Environment.Exit(-1);
        }
        /** the constructor for stack object
         */
        public Stack()
            : this(MAX)
        {
```

```
52          }
53          /** the constructor for stack object
54           */
55          public Stack(int n)
56          {
57              if (n > MAX)
58              {
59                  Console.WriteLine("Stack size must be less than " + MAX + ".");
60                  Environment.Exit(-1);
61              }
62              _s = new int[MAX];
63              _size = n;
64              _top = -1;
65              initialize();
66          }
67          /**    the function to insert new item on stack
68           */
69          public void push(int item)
70          {
71              if (_top >= _size - 1) overflowError();
72              _top++;
73              _s[_top] = item;
74          }
75          /** the function to delete an item at the top position of the stack
76           */
77          public int pop()
78          {
79              if (_top == -1) emptyError();
80              int value = _s[_top];
81              _top--;
82              return (value);
83          }
84          /** the function to get the top element of the stack
85           */
86          public int peek()
87          {
88              if (_top == -1) emptyError();
89              return (_s[_top]);
90          }
91          /** the fuction to clear an exisiting stack
92           */
93          public void reset()
94          {
95              _top = -1;
96              initialize();
97          }
98      }
99      class TestStack
100     {
101         static void Main(string[] args)
102         {
103             Stack a = new Stack();
104             Stack b = new Stack();
105
```

```
106             a.push(10);
107             a.push(20);
108             a.pop();
109             a.push(30);
110             a.push(40);
111 
112             b.push(100);
113             b.push(200);
114             b.push(300);
115             b.push(400);
116             b.pop();
117 
118             Console.WriteLine("top of stack a = " + a.peek());
119             Console.WriteLine("top of stack b = " + b.peek());
120         }
121     }
122 }
```

Lec03.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec03
{
    class TestEcho
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Type characters and <Enter> key:");
            while (true)
            {
                char ch = (char)Console.Read();
                if (ch == '\n') break;
                Console.Write(ch);
            }
            Console.WriteLine();
        }
    }
}
```

Lec04.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec04
{
    class TestEcho
    {
        static void Main(string[] args)
        {
            String buffer;
            Console.WriteLine("Type characters in a line and <Enter> key:");
            Console.WriteLine("An empty line stops this program:");
            while (true)
            {
                buffer = Console.ReadLine();
                if (buffer.Length == 0) break;
                Console.WriteLine(buffer);
            }
            Console.WriteLine("Bye !");
        }
    }
}
```

Lec05.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec05
{
    class TestIO
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello world!");

            char c;
            int i;
            double f;
            String s;

            Console.WriteLine("Type a char, an integer, a floating number and a string : ");

            String buffer = "";
            buffer = Console.ReadLine();

            char[] separators = new char[1];
            separators[0] = ' ';
            String[] st = buffer.Split(separators);
            String tmp;
            tmp = st[0];
            c = tmp.ElementAt(0);

            tmp = st[1];
            i = int.Parse(tmp);

            tmp = st[2];
            f = double.Parse(tmp);

            s = st[3];

            Console.WriteLine("c = " + c);
            Console.WriteLine("i = " + i);
            Console.WriteLine("f = " + f);
            Console.WriteLine("s = " + s);
        }
    }
}
```

Lec06.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec06
{
    /**
        This class implements LIFO list for integer values.
    */
    class Stack
    {
        // Attributes
        static int MAX = 100;
        /** an array to save stack contents
         */
        private int[] _s;
        /** the index to point top of stack
         */
        private int _top;
        /** size of the stack
         */
        private int _size;
        // Operations
        /** initialization procedure for new stack
         */
        private void initialize()
        {
            for (int i = 0; i < _size; i++)
            {
                _s[i] = 0;
            }
        }
        /** this function is called for stack overflow exception
         */
        private void overflowError()
        {
            Console.WriteLine("Stack overflow error occurs.");
            Environment.Exit(-1);
        }
        /** this function is called for stack empty exception
         */
        private void emptyError()
        {
            Console.WriteLine("Stack empty error occurs.");
            Environment.Exit(-1);
        }
        /** the default constructor for stack object
         */
        public Stack()
```

```
52              : this(MAX)
53          {
54          }
55          /** the constructor for stack object
56           */
57          public Stack(int n)
58          {
59              if (n > MAX)
60              {
61                  Console.WriteLine("Stack size must be less than " + MAX + ".");
62                  Environment.Exit(-1);
63              }
64              _s = new int[n];
65              _size = n;
66              _top = -1;
67              initialize();
68          }
69          /**    the function to insert new item on stack
70           */
71          public void push(int item)
72          {
73              if (_top >= _size - 1) overflowError();
74              _top++;
75              _s[_top] = item;
76          }
77          /** the function to delete an item at the top position of the stack
78           */
79          public int pop()
80          {
81              if (_top == -1) emptyError();
82              int value = _s[_top];
83              _top--;
84              return (value);
85          }
86          /** the function to get the top element of the stack
87           */
88          public int peek()
89          {
90              if (_top == -1) emptyError();
91              return (_s[_top]);
92          }
93          /** the fuction to clear an exisiting stack
94           */
95          public void reset()
96          {
97              _top = -1;
98              initialize();
99          }
100         /** the funtion to check if the stack is empty
101          */
102         public bool isEmpty()
103         {
104             if (_top == -1) return true;
105             else return false;
```

```
106         }
107     }
108     class LineBuffer
109     {
110         public static int ID_QUIT = 1;
111         public static int ID_PLUS = 2;
112         public static int ID_MINUS = 3;
113         public static int ID_MULTIPLY = 4;
114         public static int ID_DIVIDE = 5;
115         public static int ID_EOD = 6;
116         public static int ID_OPERAND = 7;
117 
118         static int BUFSIZ = 256;
119 
120         private int _position;
121         private char[] _text;
122         private int _tokenValue;
123         public LineBuffer(String text)
124         {
125             _text = new char[BUFSIZ];
126 
127             // copy string to char array like strcpy() in C language
128             for (int i = 0; i < text.Length; i++)
129             {
130                 _text[i] = text.ElementAt(i);
131             }
132             _text[text.Length] = '\0';
133 
134             _position = 0;
135             _tokenValue = 0;
136         }
137         public int getTokenValue()
138         {
139             return _tokenValue;
140         }
141         public int getNextToken()
142         {
143             // skip blanks
144             while (_text[_position] == ' ') _position++;
145 
146             if (_text[_position] == '\0') return ID_EOD;
147             if (_text[_position] == '+')
148             {
149                 _position++;
150                 return ID_PLUS;
151             }
152             if (_text[_position] == '*')
153             {
154                 _position++;
155                 return ID_MULTIPLY;
156             }
157             if (_text[_position] == '/')
158             {
159                 _position++;
```

```
160                 return ID_DIVIDE;
161             }
162             if ((_text[_position] == '-' && _text[_position + 1] == ' ') ||
163                 (_text[_position] == '-' && _text[_position + 1] == '\0'))
164             {
165                 _position++;
166                 return ID_MINUS;
167             }
168             String buffer = "";
169             int i = 0;
170 
171             if (_text[_position] == '-')
172             {
173                 buffer = buffer + "-";
174                 i++;
175                 _position++;
176             }
177             while (_text[_position] >= '0' && _text[_position] <= '9')
178             {
179                 buffer = buffer + _text[_position];
180                 i++;
181                 _position++;
182             }
183 
184             _tokenValue = int.Parse(buffer);
185 
186             if (_text[_position] != ' ' && _text[_position] != '\0')
187                 return ID_QUIT;
188             return ID_OPERAND;
189         }
190     }
191     class PostfixEvaluator
192     {
193         static void Main(string[] args)
194         {
195             Console.WriteLine("Type postfix expression: (ex) 1 2 3 + + ");
196             Stack operands = new Stack();
197             while (true)
198             {
199                 String aLine;
200 
201                 aLine = Console.ReadLine();
202 
203                 LineBuffer buffer = new LineBuffer(aLine);
204 
205                 while (true)
206                 {
207                     int value = 0;
208                     int tokenID = buffer.getNextToken();
209                     value = buffer.getTokenValue();
210                     if (tokenID == LineBuffer.ID_QUIT)
211                     { // "quit"
212                         Environment.Exit(0);
213                     }
```

```
214                     else if (tokenID == LineBuffer.ID_PLUS)
215                     { // operator "+"
216                         int a = operands.pop();
217                         int b = operands.pop();
218                         operands.push(a + b);
219                     }
220                     else if (tokenID == LineBuffer.ID_MINUS)
221                     { // operator "-"
222                         int a = operands.pop();
223                         int b = operands.pop();
224                         operands.push(b - a);
225                     }
226                     else if (tokenID == LineBuffer.ID_MULTIPLY)
227                     { // operator "*"
228                         int a = operands.pop();
229                         int b = operands.pop();
230                         operands.push(a * b);
231                     }
232                     else if (tokenID == LineBuffer.ID_DIVIDE)
233                     { // operator "/"
234                         int a = operands.pop();
235                         int b = operands.pop();
236                         operands.push(b / a);
237                     }
238                     else if (tokenID == LineBuffer.ID_EOD)
239                     { // end of data
240                         int data = operands.pop();
241                         if (operands.isEmpty())
242                         {
243                             Console.WriteLine("= " + data);
244                         }
245                         else
246                         {
247                             Console.WriteLine("incorrect expression");
248                         }
249                         operands.reset();
250                         break;
251                     }
252                     else
253                     { // LineBuffer.ID_OPERAND
254                         operands.push(value);
255                     }
256                 }
257             }
258         }
259     }
260 }
```

Lec07.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec07
{
    class Stack
    {
        static int MAX = 2;
        // Attributes
        /** a pointer to an array for stack contents
         */
        private int[] _s;
        /** the pointer to point top of stack
         */
        private int _top;
        /** size of the stack
         */
        private int _size;
        // Operations
        /** initialization procedure for new stack
         */
        private void initialize()
        {
            for (int i = 0; i < _size; i++)
            {
                _s[i] = 0;
            }
        }
        /** this function is called for stack empty exception
         */
        private void emptyError()
        {
            Console.WriteLine("Stack empty error occurs.");
            Environment.Exit(-1);
        }
        /** the default constructor for stack object
         */
        public Stack()
            : this(MAX)
        {
        }
        /** the constructor for stack object
         */
        public Stack(int n)
        {
            _s = new int[n];
            _size = n;
            _top = -1;
```

```
52              initialize();
53          }
54          /**    the function to insert new item on stack
55           */
56          public void push(int item)
57          {
58              if (_top >= _size - 1)
59              {
60                  int[] newS;
61                  newS = new int[2 * _size];
62                  for (int i = 0; i < _size; i++)
63                  {
64                      newS[i] = _s[i];
65                  }
66                  _s = newS;
67                  _size = 2 * _size;
68              }
69              _top++;
70              _s[_top] = item;
71          }
72          /** the function to delete an item at the top position of the stack
73           */
74          public int pop()
75          {
76              if (_top == -1) emptyError();
77              int value = _s[_top];
78              _top--;
79              return (value);
80          }
81          /** the function to get the top element of the stack
82           */
83          public int peek()
84          {
85              if (_top == -1) emptyError();
86              return (_s[_top]);
87          }
88          /** the fuction to clear an exisiting stack
89           */
90          public void reset()
91          {
92              _top = -1;
93              initialize();
94          }
95          /** the funtion to check if the stack is empty
96           */
97          public bool isEmpty()
98          {
99              if (_top == -1) return true;
100             else return false;
101         }
102     }
103     class LineBuffer
104     {
105         // Lec06에서 복사
```

```
106     }
107     class PostfixEvaluator
108     {
109         static void Main(string[] args)
110         {
111             // Lec06에서 복사
112         }
113     }
114 }
```

Lec08.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec08
{
    class Stack
    {
        public class StackItem
        {
            private int _value;
            private StackItem _next;
            public StackItem(int v)
            {
                _value = v;
                _next = null;
            }
            public int getValue()
            {
                return _value;
            }
            public StackItem getNextItem()
            {
                return _next;
            }
            public void setNextItem(StackItem item)
            {
                _next = item;
            }
        }
        // Attributes
        /** a pointer to the top item of the Stack
         */
        private StackItem _top;
        // Operations
        /** initialization procedure for new stack
         */
        void initialize()
        {
            _top = null;
        }
        /** this function is called for stack empty exception
         */
        void emptyError()
        {
            Console.WriteLine("Stack empty error occurs.");
            Environment.Exit(-1);
        }
        /** the constructor for stack object
```

```
52          */
53         public Stack()
54         {
55             initialize();
56         }
57         /**   the function to insert new item on stack
58          */
59         public void push(int i)
60         {
61             if (_top == null)
62                 _top = new StackItem(i);
63             else
64             {
65                 StackItem item = new StackItem(i);
66                 item.setNextItem(_top);
67                 _top = item;
68             }
69         }
70         /** the function to delete an item at the top position of the stack
71          */
72         public int pop()
73         {
74             if (_top == null) emptyError();
75             StackItem topItem = _top;
76             _top = _top.getNextItem();
77             return topItem.getValue();
78         }
79         /** the function to get the top element of the stack
80          */
81         public int peek()
82         {
83             if (_top == null) emptyError();
84             return (_top.getValue());
85         }
86         /** the fuction to clear an exisiting stack
87          */
88         public void reset()
89         {
90             initialize();
91         }
92         /** the funtion to check if the stack is empty
93          */
94         public bool isEmpty()
95         {
96             if (_top == null) return true;
97             else return false;
98         }
99     }
100    class LineBuffer
101    {
102        // Lec06에서 복사
103    }
104    class PostfixEvaluator
105    {
```

```
106         static void Main(string[] args)
107         {
108             // Lec06에서 복사
109         }
110     }
111 }
```

Lec09.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec09
{
    class StackException : Exception
    {
        private String _className;
        private String _functionName;
        private String _reason;
        public StackException(String cName, String fName, String reason)
        {
            _className = cName;
            _functionName = fName;
            _reason = reason;
        }
        public String getClassName()
        {
            return _className;
        }
        public String getFunctionName()
        {
            return _functionName;
        }
        public String getReason()
        {
            return _reason;
        }
    }
    class Stack
    {
        public class StackItem
        {
            private int _value;
            private StackItem _next;
            public StackItem(int v)
            {
                _value = v;
                _next = null;
            }
            public int getValue()
            {
                return _value;
            }
            public StackItem getNextItem()
            {
                return _next;
            }
```

```
52              public void setNextItem(StackItem item)
53              {
54                  _next = item;
55              }
56          }
57          // Attributes
58          /** a pointer to the top item of the Stack
59           */
60          private StackItem _top;
61          // Operations
62          /** initialization procedure for new stack
63           */
64          void initialize()
65          {
66              _top = null;
67          }
68          /** the constructor for stack object
69           */
70          public Stack()
71          {
72              initialize();
73          }
74          /**   the function to insert new item on stack
75           */
76          public void push(int i)
77          {
78              if (_top == null)
79                  _top = new StackItem(i);
80              else
81              {
82                  StackItem item = new StackItem(i);
83                  item.setNextItem(_top);
84                  _top = item;
85              }
86          }
87          /** the function to delete an item at the top position of the stack
88           */
89          public int pop()
90          {
91              if (_top == null) throw new StackException("Stack", "pop()", "Stack empty error"
);
92              StackItem topItem = _top;
93              _top = _top.getNextItem();
94              return topItem.getValue();
95          }
96          /** the function to get the top element of the stack
97           */
98          public int peek()
99          {
100             if (_top == null) throw new StackException("Stack", "peek()", "Stack empty
error");
101             return (_top.getValue());
102         }
103         /** the fuction to clear an exisiting stack
```

```
104          */
105         public void reset()
106         {
107             initialize();
108         }
109         /** the funtion to check if the stack is empty
110          */
111         public bool isEmpty()
112         {
113             if (_top == null) return true;
114             else return false;
115         }
116     }
117     class LineBuffer
118     {
119         // Lec06에서 복사
120     }
121     class Program
122     {
123         static void Main(string[] args)
124         {
125             Console.WriteLine("Type postfix expression: (ex) 1 2 3 + + ");
126             Stack operands = new Stack();
127             while (true)
128             {
129                 String aLine;
130 
131                 aLine = Console.ReadLine();
132 
133                 LineBuffer buffer = new LineBuffer(aLine);
134 
135                 while (true)
136                 {
137                     int value = 0;
138                     int tokenID = buffer.getNextToken();
139                     value = buffer.getTokenValue();
140                     try
141                     {
142                         if (tokenID == LineBuffer.ID_QUIT)
143                         { // "quit"
144                             Environment.Exit(0);
145                         }
146                         else if (tokenID == LineBuffer.ID_PLUS)
147                         { // operator "+"
148                             int a = operands.pop();
149                             int b = operands.pop();
150                             operands.push(a + b);
151                         }
152                         else if (tokenID == LineBuffer.ID_MINUS)
153                         { // operator "-"
154                             int a = operands.pop();
155                             int b = operands.pop();
156                             operands.push(b - a);
157                         }
```

```
158                         else if (tokenID == LineBuffer.ID_MULTIPLY)
159                         { // operator "*"
160                             int a = operands.pop();
161                             int b = operands.pop();
162                             operands.push(a * b);
163                         }
164                         else if (tokenID == LineBuffer.ID_DIVIDE)
165                         { // operator "/"
166                             int a = operands.pop();
167                             int b = operands.pop();
168                             operands.push(b / a);
169                         }
170                         else if (tokenID == LineBuffer.ID_EOD)
171                         { // end of data
172                             int data = operands.pop();
173                             if (operands.isEmpty())
174                             {
175                                 Console.WriteLine("= " + data);
176                             }
177                             else
178                             {
179                                 Console.WriteLine("incorrect expression");
180                             }
181                             operands.reset();
182                             break;
183                         }
184                         else
185                         { // LineBuffer.ID_OPERAND
186                             operands.push(value);
187                         }
188                     }
189                     catch (StackException ex)
190                     {
191                         Console.WriteLine("A stack exception(" + ex.getReason() + ") was
thrown by ");
192                         Console.WriteLine("the funtion " + ex.getFunctionName() + " of class
");
193                         Console.WriteLine(ex.getClassName() + ".");
194                         Console.WriteLine("The stack will be reset. Please try again.");
195                         operands.reset();
196                         break;
197                     }
198                 }
199             }
200         }
201     }
202 }
```

Lec10.cs

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7  namespace Lec10
 8  {
 9      class Stack<Type>
10      {
11          static int MAX = 100;
12          protected Type[] _s;
13          protected int _top;
14          protected int _size;
15          private void initialize()
16          {
17              for (int i = 0; i < _size; i++)
18              {
19                  _s[i] = default(Type);
20              }
21          }
22          private void overflowError()
23          {
24              Console.WriteLine("Stack Overflow Error");
25              Environment.Exit(-1);
26          }
27          private void emptyError()
28          {
29              Console.WriteLine("Stack Empty Error");
30              Environment.Exit(-1);
31          }
32          public Stack() :
33              this(MAX)
34          {
35          }
36          public Stack(int n)
37          {
38              _s = new Type[n];
39              _size = n;
40              _top = -1;
41              initialize();
42          }
43          public void push(Type item)
44          {
45              if (_top >= _size - 1) overflowError();
46              _top++;
47              _s[_top] = item;
48          }
49          public Type pop()
50          {
51              if (_top == -1) emptyError();
```

```
52              Type value = _s[_top];
53              _top--;
54              return (value);
55          }
56          public Type peek()
57          {
58              if (_top == -1) emptyError();
59              return (_s[_top]);
60          }
61          public void reset()
62          {
63              _top = -1;
64              initialize();
65          }
66          public bool isEmpty()
67          {
68              if (_top == -1) return true;
69              else return false;
70          }
71      }
72      class TestStack
73      {
74          static void Main(string[] args)
75          {
76              Stack<int> a = new Stack<int>(10);
77              Stack<int> b = new Stack<int>(20);
78              Stack<double> c = new Stack<double>(10);
79              Stack<String> d = new Stack<String>(10);
80              a.push(1);
81              a.push(2);
82              b.push(30);
83              b.push(20);
84              c.push(1.3);
85              c.push(2.4);
86              d.push("kim");
87              d.push("lee");
88              Console.WriteLine(a.pop());
89              Console.WriteLine(a.pop());
90              Console.WriteLine(b.pop());
91              Console.WriteLine(b.pop());
92              Console.WriteLine(c.pop());
93              Console.WriteLine(c.peek());
94              Console.WriteLine(d.peek());
95              Console.WriteLine(d.pop());
96              Console.WriteLine(a.isEmpty());
97              Console.WriteLine(b.isEmpty());
98              Console.WriteLine(c.isEmpty());
99              Console.WriteLine(d.isEmpty());
100         }
101     }
102 }
```

Lec11.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec11
{
    class ComplexNumber
    {
        private double m_x;
        private double m_y;

        public ComplexNumber()
        {
            m_x = 0.0;
            m_y = 0.0;
        }
        public ComplexNumber(double real, double imaginary)
        {
            m_x = real;
            m_y = imaginary;
        }
        public double real()
        {
            return m_x;
        }
        public double imaginary()
        {
            return m_y;
        }
        public double magnitude()
        {
            return Math.Sqrt(m_x * m_x + m_y * m_y);
        }
        public void set(ComplexNumber c)
        {
            m_x = c.m_x;
            m_y = c.m_y;
        }
        public ComplexNumber add(ComplexNumber c)
        {
            return new ComplexNumber(m_x + c.m_x, m_y + c.m_y);
        }
        public ComplexNumber subtract(ComplexNumber c)
        {
            return new ComplexNumber(m_x - c.m_x, m_y - c.m_y);
        }
        public ComplexNumber multiply(ComplexNumber c)
        {
            double realPart = m_x * c.m_x - m_y * c.m_y;
```

```
52              double imaginaryPart = m_x * c.m_y + m_y * c.m_x;
53
54              return new ComplexNumber(realPart, imaginaryPart);
55          }
56          public override String ToString()
57          {
58              String tmp = "";
59
60              if (m_x >= 0)
61                  tmp = tmp + (float)m_x;
62              else
63                  tmp = tmp + "(" + (float)m_x + ")";
64
65              tmp = tmp + "+";
66
67              if (m_y >= 0)
68                  tmp = tmp + (float)m_y + "i";
69              else
70                  tmp = tmp + "(" + (float)m_y + ")i";
71
72              return tmp;
73          }
74      }
75      class TestComplexNumber
76      {
77          static void Main(string[] args)
78          {
79              ComplexNumber a = new ComplexNumber(1.2, 1.5);
80              ComplexNumber b = new ComplexNumber(2.1, 3.2);
81              ComplexNumber c;
82
83              c = a.add(b);
84
85              Console.WriteLine(c.real() + "," + c.imaginary());
86              Console.WriteLine(c.magnitude());
87              Console.WriteLine("a = " + a);
88              Console.WriteLine("b = " + b);
89              Console.WriteLine("a+b = " + c);
90
91              c = a.subtract(b);
92
93              Console.WriteLine("a-b = " + c);
94
95              c = a.multiply(b);
96
97              Console.WriteLine("a*b = " + c);
98          }
99      }
100 }
```

Lec12.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec12
{
    class TestString
    {
        static void Main(string[] args)
        {
            String a = "";
            String b = new String('x',5);
            String c = "world";
            String d = b;

            Console.WriteLine(a);
            Console.WriteLine(b);
            Console.WriteLine(c);
            Console.WriteLine(d);

            b = a = "Hi, Professor Kim! ";
            c = a + b;
            d = d + " How are you ?";

            Console.WriteLine(a);
            Console.WriteLine(b);
            Console.WriteLine(c);
            Console.WriteLine(d);

            Console.WriteLine(d.Length);
        }
    }
}
```

Lec13.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec13
{
    class Test2DArray
    {
        static int sum(int[,] a)
        {
            int s = 0;
            for (int i = 0; i < a.GetLength(0); i++)
            {
                for (int j = 0; j < a.GetLength(1); j++)
                {
                    s = s + a[i, j];
                }
            }
            return s;
        }
        static int sum(int[][] a)
        {
            int s = 0;
            for (int i = 0; i < a.Length; i++)
            {
                for (int j = 0; j < a[i].Length; j++)
                {
                    s = s + a[i][j];
                }
            }
            return s;
        }
        static void Main(string[] args)
        {
            int[,] array = new int[10, 10];
            int data = 1;
            for (int i = 0; i < 10; i++)
            {
                for (int j = 0; j < 10; j++)
                {
                    array[i, j] = data;
                    data++;
                }
            }
            Console.WriteLine(sum(array));

            int[][] strangeArray;
            strangeArray = new int[3][];
            strangeArray[0] = new int[4];
```

```
52              strangeArray[1] = new int[6];
53              strangeArray[2] = new int[2];
54
55              for (int i = 0; i < strangeArray.Length; i++)
56              {
57                  for (int j = 0; j < strangeArray[i].Length; j++)
58                  {
59                      strangeArray[i][j] = j + 1;
60                  }
61              }
62              Console.WriteLine(sum(strangeArray));
63          }
64      }
65  }
```

Lec14.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec14
{
    class Matrix
    {
        protected double[][] m_ptr;
        protected int m_row;
        protected int m_col;
        protected int m_precision;
        protected void copy(Matrix m)
        {
            int i, j;

            m_row = m.m_row;
            m_col = m.m_col;
            m_precision = m.m_precision;
            m_ptr = new double[m_row][];
            for (i = 0; i < m_row; i++)
                m_ptr[i] = new double[m_col];

            for (i = 0; i < m_row; i++)
                for (j = 0; j < m_col; j++)
                    m_ptr[i][j] = m.m_ptr[i][j];
        }
        protected int getMaxDataWidth()
        {
            return 0;
        }
        public Matrix()
        {
            m_row = 0;
            m_col = 0;
            m_ptr = null;
            m_precision = 0; // 마치 정수형 인것처럼 출력
        }
        public Matrix(int row, int col)
        {
            m_row = row;
            m_col = col;
            m_precision = 0;
            m_ptr = new double[row][];
            for (int i = 0; i < row; i++)
                m_ptr[i] = new double[col];
        }
        public Matrix(Matrix m)
        {
```

```
52              copy(m);
53          }
54          public int row()
55          {
56              return m_row;
57          }
58          public int column()
59          {
60              return m_col;
61          }
62          public void setPrecision(int x)
63          {
64              m_precision = x;
65          }
66          public double get(int i, int j)
67          {
68              return m_ptr[i][j];
69          }
70          public void set(int i, int j, double v)
71          {
72              m_ptr[i][j] = v;
73          }
74          public Matrix set(Matrix m)
75          {
76              copy(m);
77              return this;
78          }
79          public Matrix add(Matrix m)
80          {
81              if (m_row != m.m_row || m_col != m.m_col)
82              {
83                  Console.WriteLine("error occurs! - size mismatch");
84                  Environment.Exit(-1);
85              }
86
87              Matrix tmp = new Matrix(m_row, m_col);
88
89              for (int i = 0; i < m_row; i++)
90                  for (int j = 0; j < m_col; j++)
91                      tmp.m_ptr[i][j] = m_ptr[i][j] + m.m_ptr[i][j];
92
93              return tmp;
94          }
95          public Matrix subtract(Matrix m)
96          {
97              // 뺄셈은 직접 만들어봐!
98              Matrix tmp = new Matrix(m_row, m_col);
99              return tmp;
100         }
101         public Matrix multiply(Matrix m)
102         {
103             if (m_col != m.m_row)
104             {
105                 Console.WriteLine("error occurs! - size mismatch for multplication");
```

```
106                 Environment.Exit(-1);
107             }
108 
109             Matrix tmp = new Matrix(m_row, m.m_col);
110 
111             for (int i = 0; i < m_row; i++)
112                 for (int j = 0; j < m.m_col; j++)
113                 {
114                     double sum = 0;
115                     for (int k = 0; k < m_col; k++)
116                         sum = sum + m_ptr[i][k] * m.m_ptr[k][j];
117                     tmp.m_ptr[i][j] = sum;
118                 }
119 
120             return tmp;
121         }
122         public Matrix multiply(double x)
123         {
124             Matrix tmp = new Matrix(m_row, m_col);
125 
126             for (int i = 0; i < m_row; i++)
127                 for (int j = 0; j < m_col; j++)
128                     tmp.m_ptr[i][j] = m_ptr[i][j] * x;
129 
130             return tmp;
131         }
132         public void readDataFromConsole()
133         {
134             int row = m_row;
135             int col = m_col;
136             double[][] p = m_ptr;
137             int i, j;
138 
139             for (i = 0; i < row; i++)
140                 for (j = 0; j < col; j++)
141                 {
142                     double x;
143                     Console.Write("data[" + i + "][" + j + "] = ");
144                     x = Double.Parse(Console.ReadLine());
145                     p[i][j] = x;
146                 }
147         }
148         public override String ToString()
149         {
150             String tmp = "";
151             int row = m_row;
152             int col = m_col;
153             double[][] p = m_ptr;
154             for (int i = 0; i < row; i++)
155             {
156                 for (int j = 0; j < col; j++)
157                 {
158                     tmp = tmp + p[i][j] + " ";
159                 }
```

```
160                 tmp = tmp + "\n";
161             }
162             return tmp;
163         }
164     }
165     class TestMatrix
166     {
167         static int readInt()
168         {
169             String s = Console.ReadLine();
170             return int.Parse(s);
171         }
172         static void Main(string[] args)
173         {
174             int m, n;
175             int i, j;
176 
177             Console.WriteLine("2 차원 배열을 두 개를 입력해 주세용:");
178             Console.Write("행 값은 얼마인가용? ");
179             m = readInt();
180             Console.Write("열 값은 얼마인가용? ");
181             n = readInt();
182             Console.WriteLine("첫번째 배열 A에 들어갈 데이터를 쳐 주세용:");
183 
184             Matrix A = new Matrix(m, n);
185 
186             for (i = 0; i < m; i++)
187                 for (j = 0; j < n; j++)
188                 {
189                     double x;
190 
191                     Console.Write("A[" + i + "][" + j + "] = ");
192                     x = readInt();
193                     A.set(i, j, x);
194                 }
195             Console.WriteLine("A = \n" + A);
196 
197             Console.WriteLine("두번째 배열 B에 들어갈 데이터를 쳐 주세용:");
198 
199             Matrix B = new Matrix(m, n);
200 
201             B.readDataFromConsole();
202             Console.WriteLine("B = \n" + B);
203 
204             Matrix X;
205 
206             X = A.add(B);
207             Console.WriteLine("A + B = \n" + X);
208 
209             // 배열의 곱도 한번 해보자.
210             // c[i][j] = a[i][0]*b[0][j] + a[i][1]*b[1][j] + ... + a[i][k]*b[k][j]
211 
212             Console.WriteLine(n + "행 3열 짜리 배열에 들어갈 데이터를 쳐 주세용:");
213 
```

```
214             Matrix C = new Matrix(n, 3);
215
216             C.readDataFromConsole();
217             Console.WriteLine("C = \n" + C);
218
219             Matrix Y;
220
221             Y = B.multiply(C);
222             Console.WriteLine("B * C = \n" + Y);
223
224             Y = B.multiply(10);
225             Console.WriteLine("B * 10 = \n" + Y);
226         }
227     }
228 }
```

Lec15.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec15
{
    class Matrix
    {
        // Lec14에서 복사
    }

    class ProductMatrix : Matrix
    {
        public ProductMatrix()
            : base()
        {

        }
        public ProductMatrix(int row, int col)
            : base(row, col)
        {
        }
        public void readDataFromConsole()
        {
            int row = m_row;
            int col = m_col;
            double[][] p = m_ptr;
            int i;

            for (i = 0; i < row; i++)
            {
                Console.WriteLine((i + 1) + "번째 사원이 ...");
                Console.Write("   냉장고는 몇대 팔았는고? ");
                p[i][0] = Double.Parse(Console.ReadLine());
                Console.Write("   에어콘은 몇대 팔았는고? ");
                p[i][1] = Double.Parse(Console.ReadLine());
                Console.Write("   선풍기는 몇대 팔았는고? ");
                p[i][2] = Double.Parse(Console.ReadLine());
            }
        }
        public override String ToString()
        {
            String tmp = "";
            int row = m_row;
            int col = m_col;
            double[][] p = m_ptr;
            int i;

            tmp = tmp + "          냉장고 에어콘 선풍기\n";
```

```
52              for (i = 0; i < row; i++)
53              {
54                  tmp = tmp + (i + 1) + "번 사원";
55                  tmp = tmp + "    " + (int)p[i][0] + "대";
56                  tmp = tmp + "    " + (int)p[i][1] + "대";
57                  tmp = tmp + "    " + (int)p[i][2] + "대";
58                  tmp = tmp + "\n";
59              }
60              tmp = tmp + "\n";
61              return tmp;
62          }
63      }
64      class TestProductMatrix
65      {
66          static void Main(string[] args)
67          {
68              int m;
69              int n = 3; // 사원별 냉장고, 에어콘, 선풍기 판매수량
70              int i;
71
72              Console.WriteLine("메트릭스 한번 사용해 봅세!!!");
73              Console.Write("사원 수는 몇명인고? ");
74              m = int.Parse(Console.ReadLine());
75              Console.WriteLine("각 사원들의 매출 수를 입력해주소.");
76
77              ProductMatrix A = new ProductMatrix(m, n);
78
79              A.readDataFromConsole();
80              Console.WriteLine("\n요렇게 팔았단 말이지!\n" + A);
81
82              Matrix B = new Matrix(n, 1);
83
84              B.set(0, 0, 100); // 냉장고 가격 100 만원
85              B.set(1, 0, 50); // 에어콘 가격 50 만원
86              B.set(2, 0, 10); // 선풍기 가격 10 만원
87
88              Matrix C;
89
90              C = A.multiply(B);
91              Console.WriteLine("그렇다면 ...");
92              for (i = 0; i < C.row(); i++)
93              {
94                  Console.Write((i + 1) + "번 사원은 ");
95                  Console.WriteLine((int)C.get(i, 0) + "만원 어치 파셨구만.");
96              }
97          }
98      }
99  }
```

Lec16.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec16
{
    class Matrix
    {
        //Lec14에서 복사
    }

    class StringArray
    {
        private String[] pArray;
        private int nArray;
        public StringArray()
        {
            nArray = 0;
            pArray = null;
        }
        public StringArray(int n)
        {
            nArray = n;
            pArray = new String[n];
        }
        public StringArray(StringArray sa)
        {
            nArray = sa.nArray;
            pArray = new String[nArray];
            for (int i = 0; i < nArray; i++)
            {
                pArray[i] = sa.pArray[i];
            }
        }
        public int size()
        {
            return nArray;
        }
        public String get(int i)
        {
            return pArray[i];
        }
        public void set(int i, String s)
        {
            pArray[i] = s;
        }
        public int getMaxStringLength()
        {
            int maxLength = 0;
```

```
52
53              for (int i = 0; i < nArray; i++)
54              {
55                  if (pArray[i].Length > maxLength)
56                      maxLength = pArray[i].Length;
57              }
58              return maxLength;
59          }
60          public void readDataFromConsole()
61          {
62              String buffer;
63
64              for (int i = 0; i < nArray; i++)
65              {
66                  Console.Write((i + 1) + "번째 이름은: ");
67                  buffer = Console.ReadLine();
68                  pArray[i] = buffer;
69              }
70          }
71          public override String ToString()
72          {
73              String tmp = "";
74              for (int i = 0; i < nArray; i++)
75              {
76                  tmp = tmp + pArray[i] + "\n";
77              }
78              return tmp;
79          }
80      }
81      class MatrixWithNames : Matrix
82      {
83          private StringArray rowNames;
84          private StringArray colNames;
85          private String unitName;
86          private String question;
87          public MatrixWithNames()
88              : base()
89          {
90              unitName = "";
91              question = "";
92              rowNames = null;
93              colNames = null;
94          }
95          public MatrixWithNames(int row, int col)
96              : base(row, col)
97          {
98              unitName = "";
99              question = "";
100             rowNames = null;
101             colNames = null;
102         }
103         public MatrixWithNames(int row, int col, StringArray rNames, StringArray cNames,
String uName, String q)
104             : base(row, col)
```

```
105         {
106             unitName = uName;
107             question = q;
108             rowNames = new StringArray(rNames);
109             colNames = new StringArray(cNames);
110         }
111         public MatrixWithNames(Matrix m, StringArray rNames, StringArray cNames, String
uName, String q)
112             : base(m)
113         {
114             unitName = uName;
115             question = q;
116             rowNames = new StringArray(rNames);
117             colNames = new StringArray(cNames);
118         }
119         public void setRowNames(StringArray names)
120         {
121             rowNames = new StringArray(names);
122         }
123         public void setColumnNames(StringArray names)
124         {
125             colNames = new StringArray(names);
126         }
127         public void setUnitName(String name)
128         {
129             unitName = name;
130         }
131         public void setQuestion(String name)
132         {
133             question = name;
134         }
135         public void readDataFromConsole()
136         {
137             int row = m_row;
138             int col = m_col;
139             double[][] p = m_ptr;
140             int i, j;
141 
142             for (i = 0; i < row; i++)
143             {
144                 Console.WriteLine(rowNames.get(i) + "의");
145                 for (j = 0; j < col; j++)
146                 {
147                     Console.Write("   " + colNames.get(j) + question + "? ");
148                     p[i][j] = Double.Parse(Console.ReadLine());
149                 }
150             }
151         }
152         public override String ToString()
153         {
154             String tmp = "";
155             int row = m_row;
156             int col = m_col;
157             double[][] p = m_ptr;
```

```
158             for (int i = 0; i < row; i++)
159             {
160                 tmp = tmp + rowNames.get(i) + " ";
161                 for (int j = 0; j < col; j++)
162                 {
163                     tmp = tmp + p[i][j] + " ";
164                 }
165                 tmp = tmp + "\n";
166             }
167             return tmp;
168         }
169     }
170
171     class TestMatrixWithNames
172     {
173         static void Main(string[] args)
174         {
175             int nProducts;
176
177             Console.Write("제품의 종류는 몇가지 ? ");
178             nProducts = int.Parse(Console.ReadLine());
179
180             StringArray productNames = new StringArray(nProducts);
181
182             Console.WriteLine("제품의 명칭을 쳐 주셔요:");
183
184             productNames.readDataFromConsole();
185
186             StringArray colNames = new StringArray(1);
187             colNames.set(0, "단가");
188             MatrixWithNames productPrices = new MatrixWithNames(nProducts, 1, productNames,
colNames, "만원", "는 얼마");
189
190             Console.WriteLine("각 제품의 단가를 쳐 주셔요:");
191
192             productPrices.readDataFromConsole();
193
194             int nClerks;
195
196             Console.Write("판매원은 몇분 ? ");
197             nClerks = int.Parse(Console.ReadLine());
198
199             StringArray clerkNames = new StringArray(nClerks);
200
201             Console.WriteLine("판매원들 성함을 쳐 주셔요:");
202
203             clerkNames.readDataFromConsole();
204
205             //    MatrixWithNames salesData = new
MatrixWithNames(nClerks,nProducts,clerkNames,productNames,"개"," 판매량은");
206             MatrixWithNames salesData = new MatrixWithNames(nClerks, nProducts);
207             salesData.setRowNames(clerkNames);
208             salesData.setColumnNames(productNames);
209             salesData.setUnitName("개");
```

```
210             salesData.setQuestion(" 판매량은");
211
212             Console.WriteLine("각 판매원들이 판매한 제품들의 수량을 쳐 주셔요:");
213
214             salesData.readDataFromConsole();
215
216             Console.WriteLine("\n지금까지 친 데이터를 확인해 드리겠습니당.");
217
218             Console.WriteLine("=== 제품 단가 ===");
219             Console.Write(productPrices);
220             Console.WriteLine("=================");
221
222             Console.WriteLine("=== 판매 자료 ===");
223             Console.Write(salesData);
224             Console.WriteLine("=================");
225
226             Matrix salesTotalPerClerk;
227
228             salesTotalPerClerk = salesData.multiply(productPrices);
229
230             Console.WriteLine("=== 결과 자료 ===");
231             Console.Write(salesTotalPerClerk);
232             Console.WriteLine("=================");
233
234             StringArray summaryNames = new StringArray(1);
235             summaryNames.set(0, "총판매액");
236             MatrixWithNames anotherSalesTotalPerClerk = new
MatrixWithNames(salesTotalPerClerk, clerkNames, summaryNames, "만원", "얼마");
237
238             Console.WriteLine("=== 최종 결과 ===");
239             Console.Write(anotherSalesTotalPerClerk);
240             Console.WriteLine("=================");
241         }
242     }
243 }
```

Lec17.cs

```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading.Tasks;
 6
 7  namespace Lec17
 8  {
 9      class Matrix
10      {
11          //Lec14에서 복사
12      }
13      class StringArray
14      {
15          //Lec16에서 복사
16      }
17      class MatrixWithNames : Matrix
18      {
19          //Lec16에서 복사
20      }
21      class TestReusability
22      {
23          static void Main(string[] args)
24          {
25          int nLectures;
26
27          Console.Write("교과목 수는 ? ");
28          nLectures = int.Parse(Console.ReadLine());
29
30          StringArray lectureNames = new StringArray(nLectures);
31
32          Console.WriteLine("교과목 명을 쳐 주셔요:");
33
34          lectureNames.readDataFromConsole();
35
36          StringArray colNames = new StringArray(1);
37          colNames.set(0,"학점");
38          MatrixWithNames lectureUnits = new
MatrixWithNames(nLectures,1,lectureNames,colNames,"학점","은 몇학점짜리");
39
40          Console.WriteLine("각 과목의 학점을 쳐 주셔요:");
41
42          lectureUnits.readDataFromConsole();
43
44          int nStudents;
45
46          Console.Write("학생은 몇명 ? ");
47          nStudents = int.Parse(Console.ReadLine());
48
49          StringArray studentNames = new StringArray(nStudents);
50
```

```
51          Console.WriteLine("학생들의 이름을 쳐 주셔요:");
52
53          studentNames.readDataFromConsole();
54
55      //    MatrixWithNames gradeData = new
MatrixWithNames(nStudents,nLectures,studentNames,lectureNames,"점"," 취득학점은");
56          MatrixWithNames gradeData = new MatrixWithNames(nStudents,nLectures);
57          gradeData.setRowNames(studentNames);
58          gradeData.setColumnNames(lectureNames);
59          gradeData.setUnitName("점");
60          gradeData.setQuestion(" 취득학점은");
61
62          Console.WriteLine("각 학생들이 취득한 학점을 쳐 주셔요:");
63
64          gradeData.readDataFromConsole();
65
66          Console.WriteLine("\n지금까지 친 데이터를 확인해 드리겠습니당.");
67
68          Console.WriteLine("=== 과목별 학점 ===");
69          Console.Write(lectureUnits);
70          Console.WriteLine("=================");
71
72          Console.WriteLine("=== 학생별 취득 학점 ===");
73          gradeData.setPrecision(2);
74          Console.Write(gradeData);
75          Console.WriteLine("========================");
76
77          Matrix gradeTotalPerStudent = new Matrix();
78
79          gradeTotalPerStudent = gradeData.multiply(lectureUnits);
80
81          gradeTotalPerStudent.setPrecision(2);
82          Console.WriteLine("=== 결과 자료 ===");
83          Console.Write(gradeTotalPerStudent);
84          Console.WriteLine("=================");
85
86          double unitsTotal = 0.0;
87
88          for(int i = 0; i < lectureUnits.row(); i++) {
89              unitsTotal = unitsTotal + lectureUnits.get(i,0);
90          }
91
92          gradeTotalPerStudent = gradeTotalPerStudent.multiply(1.0/unitsTotal);
93
94          StringArray summaryNames = new StringArray(1);
95          summaryNames.set(0,"평점평균");
96          MatrixWithNames anotherGradeTotalPerStudent = new
MatrixWithNames(gradeTotalPerStudent,studentNames,summaryNames,"점"," 취득학점은");
97
98          anotherGradeTotalPerStudent.setPrecision(4);
99
100         Console.WriteLine("=== 개인별 평균 점수 ===");
101         Console.Write(anotherGradeTotalPerStudent);
102         Console.WriteLine("========================");
```

```
103          }
104      }
105  }
```

Lec18.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec18
{
    class ListNode<Type>
    {
        public Type data;
        public ListNode<Type> pNext;
        public ListNode(Type x)
        {
            data = x;
            pNext = null;
        }
        public ListNode(Type x, ListNode<Type> next)
        {
            data = x;
            pNext = next;
        }
    }

    class LinkedList<Type>
    {
        private ListNode<Type> pHead;
        private int nCount;
        public LinkedList()
        {
            pHead = null;
            nCount = 0;
        }
        public bool isEmpty()
        {
            if (pHead == null) return true;
            else return false;
        }
        public int size()
        {
            return nCount;
        }
        public void addFirst(Type data)
        {
            ListNode<Type> pNode = new ListNode<Type>(data, pHead);
            nCount++;
            pHead = pNode;
        }
        public void addLast(Type data)
        {
            ListNode<Type> pNode = new ListNode<Type>(data);
```

```
52              nCount++;
53              if (pHead == null)
54              {
55                  pHead = pNode;
56                  return;
57              }
58              ListNode<Type> pTraverse = pHead;
59              while (pTraverse.pNext != null)
60              {
61                  pTraverse = pTraverse.pNext;
62              }
63              pTraverse.pNext = pNode;
64          }
65          public void add(int index, Type data)
66          {
67              if (index < 0 || index > nCount)
68              {
69                  Console.WriteLine("index out of bound error - add(index,data) failed.");
70                  return;
71              }
72              if (index == 0)
73              {
74                  addFirst(data);
75                  return;
76              }
77              int count = 1;
78              ListNode<Type> pFollow = pHead;
79              ListNode<Type> pTraverse = pHead.pNext;
80              while (pTraverse != null)
81              {
82                  if (index == count) break;
83                  count++;
84                  pFollow = pTraverse;
85                  pTraverse = pTraverse.pNext;
86              }
87              ListNode<Type> pNode = new ListNode<Type>(data, pTraverse);
88              nCount++;
89              pFollow.pNext = pNode;
90          }
91          public bool remove(Type data)
92          {
93              if (isEmpty() == true)
94              {
95                  Console.WriteLine("The list is empty. No data removed.");
96                  return false;
97              }
98              if (pHead != null && pHead.data.Equals(data))
99              {
100                 ListNode<Type> pNextNode = pHead.pNext;
101                 pHead = pNextNode;
102                 nCount--;
103                 return true;
104             }
105             ListNode<Type> pFollow = pHead;
```

```
106             ListNode<Type> pTraverse = pHead.pNext;
107             while (pTraverse != null)
108             {
109                 if (pTraverse.data.Equals(data))
110                 {
111                     ListNode<Type> pNextNode = pTraverse.pNext;
112                     pFollow.pNext = pNextNode;
113                     nCount--;
114                     return true;
115                 }
116                 pFollow = pTraverse;
117                 pTraverse = pTraverse.pNext;
118             }
119             Console.WriteLine(data + " is not found. No data removed.");
120             return false;
121         }
122         public ListIterator<Type> listIterator()
123         {
124             return new ListIterator<Type>(pHead);
125         }
126         public override String ToString()
127         {
128             if (isEmpty() == true)
129             {
130                 return "<>";
131             }
132             String tmp = "< ";
133             ListNode<Type> pNode = pHead;
134             while (pNode != null)
135             {
136                 tmp = tmp + pNode.data;
137                 if (pNode.pNext != null)
138                 {
139                     tmp = tmp + ", ";
140                 }
141                 else
142                 {
143                     tmp = tmp + " >";
144                 }
145                 pNode = pNode.pNext;
146             }
147             return tmp;
148         }
149     }
150 
151     class ListIterator<Type>
152     {
153         ListNode<Type> ptr;
154         public ListIterator(ListNode<Type> pHead)
155         {
156             ptr = pHead;
157         }
158         public bool hasNext()
159         {
```

```
160             if (ptr == null)
161                 return false;
162             else
163                 return true;
164         }
165         public Type next()
166         {
167             Type data = ptr.data;
168             ptr = ptr.pNext;
169             return data;
170         }
171     }
172
173     class TestLinkedList
174     {
175         static int readInt()
176         {
177             String s = Console.ReadLine();
178             return int.Parse(s);
179         }
180         static void Main(string[] args)
181         {
182             LinkedList<int> aList = new LinkedList<int>();
183             while (true)
184             {
185                 int select;
186                 Console.Write("What do you want ? <1>addFront, <2>addTail, <3>remove,
<4>quit : ");
187                 select = readInt();
188                 if (select < 1 || select > 3)
189                 {
190                     break;
191                 }
192                 Console.Write("Type data : ");
193                 int data;
194                 data = readInt();
195                 switch (select)
196                 {
197                     case 1:
198                         aList.addFirst(data);
199                         Console.WriteLine(aList);
200                         break;
201                     case 2:
202                         aList.addLast(data);
203                         Console.WriteLine(aList);
204                         break;
205                     case 3:
206                         if (aList.remove(data) == true)
207                         {
208                             Console.WriteLine(aList);
209                         }
210                         break;
211                     default:
212                         Console.WriteLine(aList);
```

```
                        break;
                }
            }
            // test other operations
            Console.WriteLine("size of the list = " + aList.size());

            Console.WriteLine("The list can be traversed like the following style.");
            ListIterator<int> i = aList.listIterator();
            while (i.hasNext())
            {
                int data = i.next();
                Console.WriteLine(data);
            }

            aList.add(2, 1000);
            Console.WriteLine(aList);
        }
    }
}
```

Lec19.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec19
{
    class ListNode<Type>
    {
        private Type data;
        private ListNode<Type> pPrev;
        private ListNode<Type> pNext;
        public ListNode()
        {
            data = default(Type);
            pPrev = this;
            pNext = this;
        }
        public ListNode(Type x)
        {
            data = x;
            pPrev = this;
            pNext = this;
        }
        public Type getData()
        {
            return data;
        }
        public void setData(Type x)
        {
            data = x;
        }
        public ListNode<Type> getNext()
        {
            return pNext;
        }
        void setNext(ListNode<Type> p)
        {
            pNext = p;
        }
        public ListNode<Type> getPrev()
        {
            return pPrev;
        }
        public void setPrev(ListNode<Type> p)
        {
            pPrev = p;
        }
        public void insert(ListNode<Type> pNode)
        {
```

```
52              // this node is inserted before pNode
53              // pNode must be understood as a chain
54              pPrev = pNode.pPrev;
55              pNext = pNode;
56              pNode.pPrev.pNext = this;
57              pNode.pPrev = this;
58          }
59          public void append(ListNode<Type> pNode)
60          {
61              // this node is appended just after pNode
62              // pNode must be understood as a chain
63              pPrev = pNode;
64              pNext = pNode.pNext;
65              pNode.pNext.pPrev = this;
66              pNode.pNext = this;
67          }
68          public void remove()
69          {
70              this.pNext.pPrev = this.pPrev;
71              this.pPrev.pNext = this.pNext;
72          }
73      };
74
75      class LinkedList<Type>
76      {
77          protected ListNode<Type> pHead;
78          protected int nCount;
79          public LinkedList()
80          {
81              pHead = null;
82              nCount = 0;
83          }
84          public bool isEmpty()
85          {
86              if (pHead == null) return true;
87              else return false;
88          }
89          public int size()
90          {
91              return nCount;
92          }
93          public void addFirst(Type data)
94          {
95              addLast(data);
96              pHead = pHead.getPrev();
97          }
98          public void addLast(Type data)
99          {
100             ListNode<Type> pNewNode = new ListNode<Type>(data);
101             nCount++;
102             if (pHead == null)
103             {
104                 pHead = pNewNode;
105                 return;
```

```
106             }
107             pNewNode.insert(pHead);
108         }
109         public void add(int index, Type data)
110         {
111             if (index < 0 || index > nCount)
112             {
113                 Console.WriteLine("index out of bound error - add(index,data) failed.");
114                 return;
115             }
116             if (index == 0)
117             {
118                 addFirst(data);
119                 return;
120             }
121             int count = 1;
122             ListNode<Type> pFollow = pHead;
123             ListNode<Type> pTraverse = pHead.getNext();
124             while (pTraverse != null)
125             {
126                 if (index == count) break;
127                 count++;
128                 pFollow = pTraverse;
129                 pTraverse = pTraverse.getNext();
130             }
131             ListNode<Type> pNewNode = new ListNode<Type>(data);
132             nCount++;
133             pNewNode.append(pFollow);
134         }
135         public bool remove(Type data)
136         {
137             if (isEmpty() == true)
138             {
139                 Console.WriteLine("The list is empty. No data removed.");
140                 return false;
141             }
142             if (pHead != null && pHead.getData().Equals(data))
143             {
144                 ListNode<Type> pNextNode = pHead.getNext();
145                 pHead.remove();
146                 nCount--;
147                 if (pNextNode == pHead) pHead = null;
148                 else pHead = pNextNode;
149                 return true;
150             }
151             ListNode<Type> pFollow = pHead;
152             ListNode<Type> pTraverse = pHead.getNext();
153             while (pTraverse != pHead)
154             {
155                 if (pTraverse.getData().Equals(data))
156                 {
157                     pTraverse.remove();
158                     nCount--;
159                     return true;
```

```
160                 }
161                 pFollow = pTraverse;
162                 pTraverse = pTraverse.getNext();
163             }
164             Console.WriteLine(data + " is not found. No data removed.");
165             return false;
166         }
167         public ListIterator<Type> listIterator()
168         {
169             return new ListIterator<Type>(pHead);
170         }
171         public override String ToString()
172         {
173             if (isEmpty() == true)
174             {
175                 return "<>";
176             }
177             String tmp = "< ";
178             ListNode<Type> pNode = pHead;
179             for (int i = 0; i < nCount; i++)
180             {
181                 tmp = tmp + pNode.getData();
182                 if (i < nCount - 1)
183                 {
184                     tmp = tmp + ", ";
185                 }
186                 else
187                 {
188                     tmp = tmp + " >";
189                 }
190                 pNode = pNode.getNext();
191             }
192             return tmp;
193         }
194     }
195 
196     class ListIterator<Type>
197     {
198         ListNode<Type> pHead;
199         ListNode<Type> ptr;
200         public ListIterator(ListNode<Type> pHead)
201         {
202             this.pHead = pHead;
203             ptr = null;
204         }
205         public bool hasNext()
206         {
207             if (ptr == null)
208             {
209                 if (pHead == null)
210                     return false;
211                 ptr = pHead;
212                 return true;
213             }
```

```
214             if (ptr == pHead)
215                 return false;
216             else
217                 return true;
218         }
219         public Type next()
220         {
221             Type data = ptr.getData();
222             ptr = ptr.getNext();
223             return data;
224         }
225     }
226     class TestLinkedList
227     {
228         //Lec18에서 복사
229     }
230 }
```

Lec20.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec20
{
    class ListNode<Type>
    {
        //Lec19에서 복사
    }

    class LinkedList<Type>
    {
        //Lec19에서 복사
    }

    class ListIterator<Type>
    {
        //Lec19에서 복사
    }

    class TestLinkedList
    {
        static int readInt()
        {
            String s = Console.ReadLine();
            return int.Parse(s);
        }
        static void Main(string[] args)
        {
            LinkedList<String> aList = new LinkedList<String>();
            while (true)
            {
                int select;
                Console.Write("What do you want ? <1>addFront, <2>addTail, <3>remove, <4>quit
: ");
                select = readInt();
                if (select < 1 || select > 3)
                {
                    break;
                }
                Console.Write("Type data : ");
                String data;
                data = Console.ReadLine();
                switch (select)
                {
                    case 1:
                        aList.addFirst(data);
                        Console.WriteLine(aList);
```

```
                        break;
                    case 2:
                        aList.addLast(data);
                        Console.WriteLine(aList);
                        break;
                    case 3:
                        if (aList.remove(data) == true)
                        {
                            Console.WriteLine(aList);
                        }
                        break;
                    default:
                        Console.WriteLine(aList);
                        break;
                }
            }
            // test other operations
            Console.WriteLine("size of the list = " + aList.size());

            Console.WriteLine("The list can be traversed like the following style.");
            ListIterator<String> i = aList.listIterator();
            while (i.hasNext())
            {
                String data = i.next();
                Console.WriteLine(data);
            }

            aList.add(2, "hello");
            Console.WriteLine(aList);
        }
    }
}
```

Lec21.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec21
{
    class ListNode<Type>
    {
        //Lec19에서 복사
    }

    class LinkedList<Type>
    {
        //Lec19에서 복사
    }

    class ListIterator<Type>
    {
        //Lec19에서 복사
    }

    class Revolver : LinkedList<String>
    {
        static bool KILLED = true;
        static bool ALIVE = false;
        private ListNode<String> pCurrent;
        private int nHoles;
        private int nBullets;
        public Revolver(bool[] bullets, int n)
        {
            nHoles = n;
            nBullets = 0;
            for (int i = 0; i < n; i++)
            {
                if (bullets[i] == true)
                {
                    addLast("*");
                    nBullets++;
                    nHoles--;
                }
                else
                {
                    addLast("0");
                }
            }
            pCurrent = pHead;
        }
        public bool bang()
        {
```

```
52              Random rand = new Random();
53              int number = rand.Next();
54              number = number - number / 10 * 10;
55              if (number % 2 == 0)
56              {
57                  for (int i = 0; i < number; i++)
58                  {
59                      pCurrent = pCurrent.getPrev();
60                  }
61                  Console.WriteLine("Rotate right " + number + " times.");
62              }
63              else
64              {
65                  for (int i = 0; i < number; i++)
66                  {
67                      pCurrent = pCurrent.getNext();
68                  }
69                  Console.WriteLine("Rotate left " + number + " times.");
70              }
71              if (pCurrent.getData().Equals("*"))
72              {
73                  Console.WriteLine("Sorry! You are dead.");
74                  nBullets--;
75                  nHoles++;
76                  return KILLED;
77              }
78              else
79              {
80                  return ALIVE;
81              }
82          }
83          public void print()
84          {
85              print(false);
86          }
87          public void print(bool first)
88          {
89              ListNode<String> tmp = pCurrent;
90              if (tmp.getData().Equals("*"))
91              {
92                  if (first == true)
93                  {
94                      Console.Write("* ");
95                  }
96                  else
97                  {
98                      Console.Write("X ");
99                      tmp.setData("0");
100                 }
101             }
102             else
103             {
104                 Console.Write("0 ");
105             }
```

```
106             tmp = tmp.getNext();
107             for (int i = 1; i < nCount; i++)
108             {
109                 Console.Write(tmp.getData() + " ");
110                 tmp = tmp.getNext();
111             }
112             Console.WriteLine();
113         }
114     }
115     class RussianRoulette
116     {
117         static void Main(string[] args)
118         {
119             int MAX = 8;
120             Console.WriteLine("--- Welcome to Hell (Russian Roulette) ---");
121             Console.WriteLine("Load Bullets : You can load up to " + MAX + " bullets.");
122             bool[] bullets = new bool[MAX];
123             int i;
124             for (i = 0; i < MAX; i++)
125             {
126                 String c;
127 
128                 Console.Write("Do you want to load a bullet at hole[" + i + "] ? (y/n) ");
129                 c = Console.ReadLine();
130                 if (c.Equals("y"))
131                 {
132                     bullets[i] = true;
133                 }
134                 else
135                 {
136                     bullets[i] = false;
137                 }
138             }
139             Console.Write("Type the number of players: ");
140             int num;
141 
142             num = int.Parse(Console.ReadLine());
143 
144             Revolver aGun = new Revolver(bullets, MAX);
145             Console.Write("Loaded bullets: ");
146             aGun.print(true);
147             i = 1;
148             while (true)
149             {
150                 String c;
151 
152                 if (i > num)
153                 {
154                     i = 1;
155                 }
156                 Console.Write(i + "th man's turn : do you want to continue ? (y/n) ");
157                 c = Console.ReadLine();
158                 if (!c.Equals("y")) Environment.Exit(0);
159                 if (aGun.bang()) Console.Beep();
```

```
160                 aGun.print();
161                 i++;
162             }
163         }
164     }
165 }
```

Lec22.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec22
{
    /*
        class Student
        {
            private String ms_name;
            private bool mb_male;
            private bool mb_rich;
            private bool mb_tall;
            private bool mb_cute;
            public Student()
            {
                ms_name = "";
                mb_male = mb_rich = mb_tall = mb_cute = false;
            }
            public Student(String s)
            {
                ms_name = s;
                mb_male = mb_rich = mb_tall = mb_cute = false;
            }
            public Student(String s, bool male, bool rich, bool tall, bool cute)
            {
                ms_name = s;
                setMale(male);
                setRich(rich);
                setTall(tall);
                setCute(cute);
            }
            public String getName()
            {
                return ms_name;
            }
            public void setMale(bool flag)
            {
                mb_male = flag;
            }
            public void setRich(bool flag)
            {
                mb_rich = flag;
            }
            public void setTall(bool flag)
            {
                mb_tall = flag;
            }
            public void setCute(bool flag)
```

```
52              {
53                  mb_cute = flag;
54              }
55              public bool isMale()
56              {
57                  return mb_male;
58              }
59              public bool isRich()
60              {
61                  return mb_rich;
62              }
63              public bool isTall()
64              {
65                  return mb_tall;
66              }
67              public bool isCute()
68              {
69                  return mb_cute;
70              }
71              public override String ToString()
72              {
73                  String tmp = "";
74                  tmp = tmp + ms_name + " is ";
75                  if (isMale()) tmp = tmp + "a boy and he is ";
76                  else tmp = tmp + "a girl and she is ";
77                  if (isRich()) tmp = tmp + "rich, ";
78                  else tmp = tmp + "poor, ";
79                  if (isTall()) tmp = tmp + "tall and ";
80                  else tmp = tmp + "short and ";
81                  if (isCute()) tmp = tmp + "cute.";
82                  else tmp = tmp + "ugly.";
83                  return tmp;
84              }
85              public void readDataFromConsole()
86              {
87                  String buffer = Console.ReadLine();
88                  char[] delemeter = new char[1];
89                  delemeter[0] = ' ';
90                  String[] data = buffer.Split(delemeter);
91                  ms_name = data[0];
92                  if (data[1].Equals("1")) setMale(true);
93                  else setMale(false);
94                  if (data[2].Equals("1")) setRich(true);
95                  else setRich(false);
96                  if (data[3].Equals("1")) setTall(true);
97                  else setTall(false);
98                  if (data[4].Equals("1")) setCute(true);
99                  else setCute(false);
100             }
101         }
102     */
103     class Student
104     {
105         static int MALE_MASK = 1 << 0;
```

```
106         static int RICH_MASK = 1 << 1;
107         static int TALL_MASK = 1 << 2;
108         static int CUTE_MASK = 1 << 3;
109         private String ms_name;
110         private int m_data;
111         public Student()
112         {
113             ms_name = "";
114             m_data = 0;
115         }
116         public Student(String s)
117         {
118             ms_name = s;
119             m_data = 0;
120         }
121         public Student(String s, bool male, bool rich, bool tall, bool cute)
122         {
123             ms_name = s;
124             setMale(male);
125             setRich(rich);
126             setTall(tall);
127             setCute(cute);
128         }
129         public String getName()
130         {
131             return ms_name;
132         }
133         public void setMale(bool flag)
134         {
135             if (flag) m_data = m_data ¦ MALE_MASK;
136             else m_data = m_data & ~MALE_MASK;
137         }
138         public void setRich(bool flag)
139         {
140             if (flag) m_data = m_data ¦ RICH_MASK;
141             else m_data = m_data & ~RICH_MASK;
142         }
143         public void setTall(bool flag)
144         {
145             if (flag) m_data = m_data ¦ TALL_MASK;
146             else m_data = m_data & ~TALL_MASK;
147         }
148         public void setCute(bool flag)
149         {
150             if (flag) m_data = m_data ¦ CUTE_MASK;
151             else m_data = m_data & ~CUTE_MASK;
152         }
153         public bool isMale()
154         {
155             if ((m_data & MALE_MASK) != 0) return true;
156             else return false;
157         }
158         public bool isRich()
159         {
```

```
160             if ((m_data & RICH_MASK) != 0) return true;
161             else return false;
162         }
163         public bool isTall()
164         {
165             if ((m_data & TALL_MASK) != 0) return true;
166             else return false;
167         }
168         public bool isCute()
169         {
170             if ((m_data & CUTE_MASK) != 0) return true;
171             else return false;
172         }
173         public override String ToString()
174         {
175             String tmp = "";
176             tmp = tmp + ms_name + " is ";
177             if (isMale()) tmp = tmp + "a boy and he is ";
178             else tmp = tmp + "a girl and she is ";
179             if (isRich()) tmp = tmp + "rich, ";
180             else tmp = tmp + "poor, ";
181             if (isTall()) tmp = tmp + "tall and ";
182             else tmp = tmp + "short and ";
183             if (isCute()) tmp = tmp + "cute.";
184             else tmp = tmp + "ugly.";
185             return tmp;
186         }
187         public void readDataFromConsole()
188         {
189             String buffer = Console.ReadLine();
190             char[] delemeter = new char[1];
191             delemeter[0] = ' ';
192             String[] data = buffer.Split(delemeter);
193 
194             ms_name = data[0];
195 
196             if (data[1].Equals("1")) setMale(true);
197             else setMale(false);
198             if (data[2].Equals("1")) setRich(true);
199             else setRich(false);
200             if (data[3].Equals("1")) setTall(true);
201             else setTall(false);
202             if (data[4].Equals("1")) setCute(true);
203             else setCute(false);
204         }
205     }
206 
207     class Program
208     {
209         static void Main(string[] args)
210         {
211             Student x = new Student("kim");
212             Student y = new Student("lee", true, false, true, true);
213             Student z = new Student("park", false, false, true, false);
```

```
214
215             x.setRich(true);
216             x.setCute(true);
217
218             y.setMale(false);
219             y.setRich(true);
220
221             z.setTall(false);
222             z.setCute(true);
223
224             Console.WriteLine(x);
225             Console.WriteLine(y);
226             Console.WriteLine(z);
227
228             Student[] st = new Student[5];
229
230             Console.WriteLine("Type information for 5 students as \"park 0 1 0 1\"");
231
232             int i;
233             for (i = 0; i < 5; i++)
234             {
235                 st[i] = new Student();
236                 st[i].readDataFromConsole();
237             }
238
239             Console.WriteLine("== favorite spouse candidates list ==");
240             for (i = 0; i < 5; i++)
241             {
242                 if (st[i].isRich() && st[i].isCute())
243                 {
244                     Console.WriteLine(st[i].getName());
245                 }
246             }
247         }
248     }
249 }
```

Lec23.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec23
{
    class ListNode<Type>
    {
        //Lec18에서 복사
    }

    class LinkedList<Type>
    {
        //Lec18에서 복사
    }

    class ListIterator<Type>
    {
        //Lec18에서 복사
    }

    abstract class MuckBBang
    {
        public static int ID_MUCKBBANG = 0;
        public static int ID_SICKBBANG = 1;
        public static int ID_MOOLBBANG = 2;
        public static int ID_SOOLBBANG = 3;

        static int BBANG_WEIGHT = 10;
        static int INIT_WEIGHT = 10000;

        protected static Random rand = new Random();

        protected int weight;
        protected String id;
        public MuckBBang(String name)
        {
            weight = INIT_WEIGHT;
            id = name;
        }


        public virtual void eatAppetizer()
        {
            Console.Write(id + "는 ");
        }

        public virtual void eatEntree(int n)
        {
```

```
52              weight = weight + n * BBANG_WEIGHT;
53          }
54
55          public virtual void eatDessert()
56          {
57              Console.WriteLine("커피 한잔으로 마무리 했음니다.");
58          }
59
60          public virtual void digest()
61          {
62              Console.WriteLine(id + "는 운동을 해서 소화를 시킵니다.");
63              addWeight(-10);
64          }
65          public virtual void addWeight(int w) { weight = weight + w; }
66          public virtual int howManyDoYouWant() { return 0; }
67          public virtual int whoAreYou() { return ID_MUCKBBANG; }
68          public abstract void print();
69      }
70
71      class MoolBBang : MuckBBang
72      {
73          private int nCup;
74
75          public MoolBBang(String name)
76              : base(name)
77          {
78              nCup = 0;
79          }
80
81          public override void eatAppetizer()
82          {
83              base.eatAppetizer();
84              nCup = nCup + 1;
85              Console.Write("에피타이저로 물 한잔 마시고, ");
86          }
87
88          public override void eatEntree(int n)
89          {
90              Console.Write("본격적으로 물을 " + n + "잔 마신 후, ");
91              nCup = nCup + n;
92          }
93
94          public override void eatDessert()
95          {
96              Console.Write("디저트로 물 두잔 마신 다음 ");
97              base.eatDessert();
98          }
99
100         public override int howManyDoYouWant()
101         {
102             int n = rand.Next();
103
104             return n % 5;
105         }
```

```
106
107         public override void print()
108         {
109             Console.WriteLine("물빵인 " + id + "는 몸무게가 " + weight + "그램이고 물을 " +
nCup + "잔 마셨읍니다.");
110         }
111
112         public override int whoAreYou() { return ID_MOOLBBANG; }
113     }
114     class SickBBang : MuckBBang
115     {
116         private int nBBang;
117         public SickBBang(String name)
118             : base(name)
119         {
120             nBBang = 0;
121         }
122
123         public override void eatAppetizer()
124         {
125             base.eatAppetizer();
126             nBBang = nBBang + 1;
127             Console.Write("에피타이저로 빵 하나 먹고, ");
128         }
129
130         public override void eatEntree(int n)
131         {
132             base.eatEntree(n);
133             Console.Write("본격적으로 빵을 " + n + "개 먹은 후, ");
134             nBBang = nBBang + n;
135         }
136
137         public override void eatDessert()
138         {
139             Console.WriteLine("디저트로 빵 세개 먹고 마무리 했읍니다.");
140         }
141
142         public override int howManyDoYouWant()
143         {
144             int n = rand.Next();
145
146             return n % 10;
147         }
148
149         public override void digest()
150         {
151             Console.WriteLine(id + "는 잠을 자면서 소화를 시킵니다.");
152             addWeight(-1);
153         }
154
155         public override void print()
156         {
157             Console.WriteLine("식빵인 " + id + "는 몸무게가 " + weight + "그램이고 빵을 " +
nBBang + "개 먹었읍니다.");
```

```
158         }
159         public override int whoAreYou() { return ID_SICKBBANG; }
160     }
161     class SoolBBang : MuckBBang
162     {
163         private int nBeer;
164         private double ccNoranmool;
165         public SoolBBang(String name)
166             : base(name)
167         {
168             nBeer = 0;
169             ccNoranmool = 0.0;
170         }
171 
172         public override void eatAppetizer()
173         {
174             base.eatAppetizer();
175             nBeer = nBeer + 10;
176             Console.Write("에피타이저로 맥주 열잔 하고, ");
177         }
178 
179         public override void eatEntree(int n)
180         {
181             int r = rand.Next();
182 
183             r = r % 1000;
184             double x = r / 2.0;
185             Console.Write("본격적으로 노란물을 " + x + "cc 마신 후, ");
186             ccNoranmool = ccNoranmool + x;
187         }
188 
189         public override int howManyDoYouWant()
190         {
191             // 미친쇄희... 1000잔이나 달라고...
192             return 1000;
193         }
194 
195         public override void digest()
196         {
197             Console.WriteLine(id + "는 오바이트하면서 소화를 시킵니다.");
198             addWeight(-300);
199         }
200 
201         public override void print()
202         {
203             Console.WriteLine("술빵인 " + id + "는 몸무게가 " + weight + "그램이고 맥주를 "
204                 + nBeer + "잔, 노란물을 " + ccNoranmool + "cc 먹었읍니다.");
205         }
206         public override int whoAreYou() { return ID_SOOLBBANG; }
207     }
208     class Servant
209     {
210         static int MAX_BBANG = 1000000;
211         private int nBBang;
```

```
212
213         public Servant()
214         {
215             nBBang = MAX_BBANG;
216         }
217
218         public int askAmountToServe(MuckBBang person)
219         {
220             int n = person.whoAreYou();
221
222             // 이안세한테는 물어보고 자시고 할 거 없음...
223             if (n == MuckBBang.ID_SOOLBBANG) return 0;
224
225             int amount = person.howManyDoYouWant();
226
227             if (n == MuckBBang.ID_MOOLBBANG) return amount;
228
229             if (amount > nBBang)
230             {
231                 amount = nBBang;
232                 nBBang = 0;
233                 return amount;
234             }
235
236             nBBang = nBBang - amount;
237             return amount;
238         }
239
240         public int getNBBang() { return nBBang; }
241     }
242
243     class DiningTable
244     {
245         private int nCustomer;
246         private LinkedList<MuckBBang> pCustomers;
247
248         public DiningTable()
249         {
250             pCustomers = new LinkedList<MuckBBang>();
251         }
252
253         public void join(MuckBBang person)
254         {
255             nCustomer++;
256             pCustomers.addLast(person);
257         }
258
259         public void servedBy(Servant pServant)
260         {
261             int i = 0;
262             ListIterator<MuckBBang> li = pCustomers.listIterator();
263             while (li.hasNext())
264             {
265                 MuckBBang person = li.next();
```

```
266
267                 person.eatAppetizer();
268                 int n = pServant.askAmountToServe(person);
269                 person.eatEntree(n);
270                 person.eatDessert();
271             }
272         }
273
274         public void print()
275         {
276             int nCustomer = pCustomers.size();
277             Console.WriteLine("테이블에는 손님이 " + nCustomer + "분 계시는데요...\n");
278             ListIterator<MuckBBang> li = pCustomers.listIterator();
279             while (li.hasNext())
280             {
281                 MuckBBang person = li.next();
282                 person.print();
283             }
284         }
285         public LinkedList<MuckBBang> getCustomers() { return pCustomers; }
286     }
287     class BBangHouse
288     {
289         static int MAX_MEAL = 10;
290         private Servant pServant;
291         private DiningTable table;
292         public BBangHouse()
293         {
294             pServant = new Servant();
295             table = new DiningTable();
296         }
297
298         public void join(MuckBBang person)
299         {
300             table.join(person);
301         }
302
303         public void run()
304         {
305             for (int meal = 0; meal < MAX_MEAL; meal++)
306             {
307                 table.servedBy(pServant); // pServant.serve(table);
308
309                 LinkedList<MuckBBang> customers = table.getCustomers();
310                 ListIterator<MuckBBang> li = customers.listIterator();
311                 while (li.hasNext())
312                 {
313                     MuckBBang person = li.next();
314                     person.digest();
315                 }
316                 Console.WriteLine("...");
317             }
318         }
319
```

```
320         public void print()
321         {
322             Console.WriteLine("=== 최종 상태 보고서 ===");
323             int nBBang = pServant.getNBBang();
324             Console.WriteLine("현재 빵집에는 빵이 " + nBBang + "개 남아있고요...");
325             table.print();
326         }
327     }
328     class BBangNara
329     {
330         static void Main(string[] args)
331         {
332             BBangHouse pHouse = new BBangHouse();
333 
334             Random rand = new Random();
335             for (int i = 0; i < 10; i++)
336             {
337                 String buffer;
338                 int id = rand.Next();
339 
340                 int category = (id % 3) + 1;
341                 id = id % 100;
342                 buffer = "ID" + id;
343                 MuckBBang pBBang = null;
344 
345                 switch (category)
346                 {
347                     case 1:
348                         pBBang = new SickBBang(buffer);
349                         break;
350                     case 2:
351                         pBBang = new MoolBBang(buffer);
352                         break;
353                     case 3:
354                         pBBang = new SoolBBang(buffer);
355                         break;
356                 }
357                 pHouse.join(pBBang);
358             }
359             pHouse.run();
360             pHouse.print();
361         }
362     }
363 }
```

Lec24.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec24
{
    interface StackItem
    {
        StackItem getNextItem();
        void setNextItem(StackItem item);
    }

    class Stack
    {
        StackItem top;
        public Stack()
        {
            top = null;
        }
        public void push(StackItem item)
        {
            if (top == null) top = item;
            else
            {
                item.setNextItem(top);
                top = item;
            }
        }
        public StackItem pop()
        {
            if (top == null)
            {
                Console.WriteLine("stack is empty");
                Environment.Exit(-1);
            }
            StackItem topItem = top;
            top = top.getNextItem();
            return topItem;
        }
        public void printAll()
        {
            Console.Write("This stack has : ");
            StackItem item = top;
            while (item != null)
            {
                Console.Write(item + " ");
                item = item.getNextItem();
            }
            Console.WriteLine();
```

```
52          }
53      }
54
55      class DefaultStackItem : StackItem
56      {
57          protected StackItem next;
58          public StackItem getNextItem()
59          {
60              return next;
61          }
62          public void setNextItem(StackItem item)
63          {
64              next = item;
65          }
66      }
67
68      class IntItem : DefaultStackItem
69      {
70          int i;
71          public IntItem(int i)
72          {
73              this.i = i; next = null;
74          }
75          public override String ToString()
76          {
77              return i + "";
78          }
79      }
80
81      class StringItem : DefaultStackItem
82      {
83          String s;
84          public StringItem(String s)
85          {
86              this.s = s; next = null;
87          }
88          public override String ToString()
89          {
90              return s;
91          }
92      }
93
94      class ComplexNumberItem : DefaultStackItem
95      {
96          double real;
97          double imaginary;
98          public ComplexNumberItem(double r, double i)
99          {
100             real = r; imaginary = i; next = null;
101         }
102         public override String ToString()
103         {
104             return real + "+" + imaginary + "i";
105         }
```

```
106     }
107
108     class StackTest
109     {
110         static void Main(string[] args)
111         {
112             Stack aStack = new Stack();
113             aStack.push(new IntItem(10));
114             aStack.push(new StringItem("kim"));
115             aStack.push(new ComplexNumberItem(1.5, 5.9));
116             aStack.push(new ComplexNumberItem(2.4, 7.1));
117             aStack.push(new StringItem("lee"));
118             aStack.push(new IntItem(9));
119             Console.WriteLine("Item removed : " + aStack.pop());
120             Console.WriteLine("Item removed : " + aStack.pop());
121             Console.WriteLine("Item removed : " + aStack.pop());
122             aStack.printAll();
123         }
124     }
125 }
```

Lec25.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lec25
{
    class ListNode<Type>
    {
        //Lec19에서 복사
    }

    class LinkedList<Type>
    {
        //Lec19에서 복사
    }

    class ListIterator<Type>
    {
        //Lec19에서 복사
    }

    class IntStack
    {
        static int MAX = 100;
        // Attributes
        private int[] s;
        private int top;
        private int arraySize;
        // Operations
        private void emptyError()
        {
            Console.WriteLine("IntStack empty error occurs.");
            Environment.Exit(-1);
        }
        public IntStack()
            : this(MAX)
        {
        }
        public IntStack(int n)
        {
            s = new int[n];
            arraySize = n;
            top = -1;
        }
        public void push(int item)
        {
            if (top >= (arraySize - 1))
            {
                int[] newS;
```

```
52                  newS = new int[2 * arraySize];
53                  for (int i = 0; i < arraySize; i++)
54                  {
55                      newS[i] = s[i];
56                  }
57                  s = newS;
58                  arraySize = 2 * arraySize;
59              }
60              top++;
61              s[top] = item;
62          }
63          public int pop()
64          {
65              if (top == -1) emptyError();
66              int value = s[top];
67              top--;
68              return (value);
69          }
70          public int peek()
71          {
72              if (top == -1) emptyError();
73              return s[top];
74          }
75          public void reset()
76          {
77              top = -1;
78          }
79          public bool isEmpty()
80          {
81              if (top == -1) return true;
82              else return false;
83          }
84          public int size()
85          {
86              if (top == -1) return 0;
87              else return top + 1;
88          }
89          public int getAt(int i)
90          {
91              return s[i];
92          }
93      }
94      class TreeNode
95      {
96          private String data;
97          private LinkedList<TreeNode> children;
98          public TreeNode(String s)
99          {
100             data = s;
101             children = new LinkedList<TreeNode>();
102         }
103         public String getData()
104         {
105             return data;
```

```
106         }
107         public void depthFirstTraverse()
108         {
109             Console.WriteLine(data);
110 
111             ListIterator<TreeNode> i = children.listIterator();
112             while (i.hasNext())
113             {
114                 TreeNode child = i.next();
115                 child.depthFirstTraverse();
116             }
117         }
118         public void depthFirstEnumeration(LinkedList<String> pEnumeration)
119         {
120             pEnumeration.addLast(data);
121             ListIterator<TreeNode> i = children.listIterator();
122             while (i.hasNext())
123             {
124                 TreeNode child = i.next();
125                 child.depthFirstEnumeration(pEnumeration);
126             }
127         }
128         public TreeNode find(String s)
129         {
130             if (data.Equals(s)) return this;
131 
132             ListIterator<TreeNode> i = children.listIterator();
133             while (i.hasNext())
134             {
135                 TreeNode child = i.next();
136                 TreeNode foundNode = child.find(s);
137                 if (foundNode != null) return foundNode;
138             }
139             return null;
140         }
141         public void addChild(String s)
142         {
143             TreeNode pNewNode = new TreeNode(s);
144             children.addLast(pNewNode);
145         }
146         public int showTree(bool isFirstChild, int level, int xPos, int gap, IntStack
pStack)
147         {
148             if (isFirstChild == true && level > 0)
149             {
150                 Console.Write("-+-");
151             }
152             else if (level > 0)
153             {
154                 Console.Write(" +-");
155             }
156             Console.Write(data);
157             if (level == 0)
158             {
```

```
159                 xPos = xPos + data.Length;
160                 pStack.push(data.Length + 1);
161             }
162             else if (children.size() > 1)
163             {
164                 xPos = xPos + data.Length + 3;
165                 pStack.push(data.Length + 2 + gap);
166             }
167             else
168             {
169                 xPos = xPos + data.Length;
170                 pStack.push(data.Length + 1 + gap);
171             }
172             int maxWidth = 0;
173             bool firstNodeFlag1 = true;
174             bool firstNodeFlag2 = true;
175             bool alreadyPoped = false;
176             int nCount = children.size();
177             ListIterator<TreeNode> iterator = children.listIterator();
178             while (iterator.hasNext())
179             {
180                 int childWidth;
181                 TreeNode child = iterator.next();
182                 nCount--;
183                 if (firstNodeFlag1 == true)
184                 {
185                     firstNodeFlag1 = false;
186                 }
187                 else
188                 {
189                     int i;
190                     for (i = 0; i < pStack.size(); i++)
191                     {
192                         int n = pStack.getAt(i);
193                         for (int j = 0; j < n; j++)
194                         {
195                             Console.Write(" ");
196                         }
197                         Console.Write("¦");
198                     }
199                     Console.WriteLine();
200                     int nBlanks = xPos;
201                     for (i = 0; i < pStack.size() - 1; i++)
202                     {
203                         int n = pStack.getAt(i);
204                         for (int j = 0; j < n; j++)
205                         {
206                             Console.Write(" ");
207                             nBlanks--;
208                         }
209                         Console.Write("¦");
210                         nBlanks--;
211                     }
212                     for (i = 0; i < nBlanks; i++)
```

```
213                     {
214                         Console.Write(" ");
215                     }
216                 }
217                 int moreGap = 0;
218                 if (child.children.size() > 1 && nCount == 0)
219                 {
220                     moreGap = pStack.pop() + 1;
221                     alreadyPoped = true;
222                 }
223                 childWidth = child.showTree(firstNodeFlag2, level + 1, xPos, moreGap,
pStack);
224                 firstNodeFlag2 = false;
225                 if (maxWidth < childWidth) maxWidth = childWidth;
226             }
227             if (firstNodeFlag1) Console.WriteLine();
228             if (alreadyPoped == false) pStack.pop();
229             return maxWidth;
230         }
231     }
232     class Tree
233     {
234         private TreeNode root;
235         public Tree()
236         {
237             root = null;
238         }
239         public void setRoot(String s)
240         {
241             root = new TreeNode(s);
242         }
243         public void depthFirstTraverse()
244         {
245             if (root == null)
246             {
247                 Console.WriteLine("No node to visit");
248                 return;
249             }
250             root.depthFirstTraverse();
251         }
252         public LinkedList<String> depthFirstEnumeration()
253         {
254             if (root == null) return null;
255             LinkedList<String> pEnumeration = new LinkedList<String>();
256             root.depthFirstEnumeration(pEnumeration);
257             return pEnumeration;
258         }
259         public bool addNewChild(String parent, String child)
260         {
261             if (root == null) return false;
262             TreeNode pNode = root.find(parent);
263             if (pNode == null) return false;
264             pNode.addChild(child);
265             return true;
```

```
266         }
267         public void showTree()
268         {
269             if (root == null)
270             {
271                 Console.WriteLine("No data in the tree!");
272                 return;
273             }
274             IntStack pStack = new IntStack();
275             root.showTree(true, 0, 0, 0, pStack);
276         }
277     }
278     class TestTree
279     {
280         static void Main(string[] args)
281         {
282             Tree aTree = new Tree();
283             String rootName;
284             Console.Write("Type the name of the root node of this tree: ");
285             rootName = Console.ReadLine();
286             aTree.setRoot(rootName);
287             while (true)
288             {
289                 String parent, child;
290 
291                 Console.WriteLine(">>");
292                 Console.Write("Type a parent name: ");
293                 parent = Console.ReadLine();
294                 Console.Write("Type a child name: ");
295                 child = Console.ReadLine();
296                 if (aTree.addNewChild(parent, child) == false) break;
297             }
298 
299             Console.WriteLine("- depth first traverse -");
300             aTree.depthFirstTraverse();
301 
302             Console.WriteLine("- depth first traverse using enumeration -");
303             LinkedList<String> pList = aTree.depthFirstEnumeration();
304 
305             ListIterator<String> i = pList.listIterator();
306             while (i.hasNext())
307             {
308                 String s = i.next();
309                 Console.WriteLine(s);
310             }
311 
312             Console.WriteLine("=== the tree you made looks like as follows ===");
313             aTree.showTree();
314         }
315     }
316 }
```