Part 04

프로젝트

Chapter 13

도서 관리 프로그램

Chapter 13

도서 관리 프로그램

학습목표

▸ 지금까지 배운 내용을 모두 활용해 프로그램을 만든다.

▸ 윈도 폼 프로젝트로 프로그램을 만드는 과정을 이해한다.

Preview

지금까지 배운 내용을 모두 종합해서 도서 관리 프로그램(/13장/BookManager)을 만들어보겠습니다. 코드가 굉장히 길지만, 지금까지 배운 내용을 합쳐놓은 것뿐입니다. 어려운 내용이 딱히 없으므로 정리하는 마음으로 가볍게 코드를 읽어봅시다.

01 동작 확인

도서 관리 프로그램을 처음 실행하면 다음과 같은 모습입니다. 도서 현황과 사용자 현황이 나오며, 도서 현황에서 도서를 클릭하고 사용자 현황에서 사용자를 클릭하면 [대여/반납] 그룹 박스 내부에 도서와 사용자의 정보가 출력됩니다.

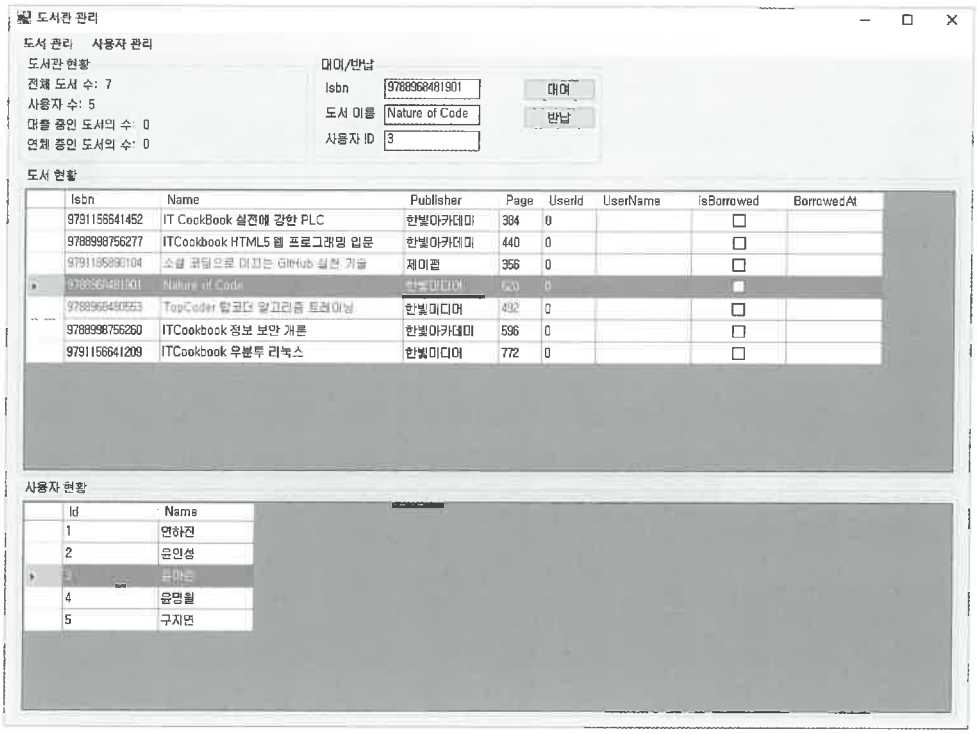

그림 13-1 도서 관리 프로그램

실제 도서관에서는 도서의 바코드와 사용자의 카드 등을 읽는 장비를 활용해서도 데이터를 넣을 수 있게 만들어주면 좋겠죠?

어쨌든 이때 [대여] 버튼을 누르면 메시지 상자가 [그림 13-2]와 같이 뜹니다. 만약 대여 중인 도서라면 "이미 대여 중인 도서입니다"라는 메시지 상자가 출력됩니다.

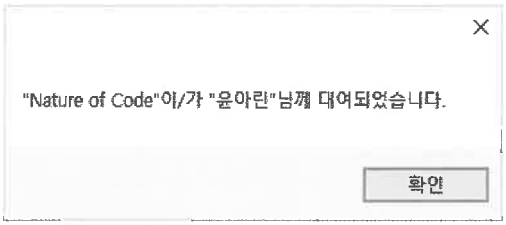

그림 13-2 대여

대여된 도서는 다음과 같이 관련 정보가 추가되어 출력됩니다.

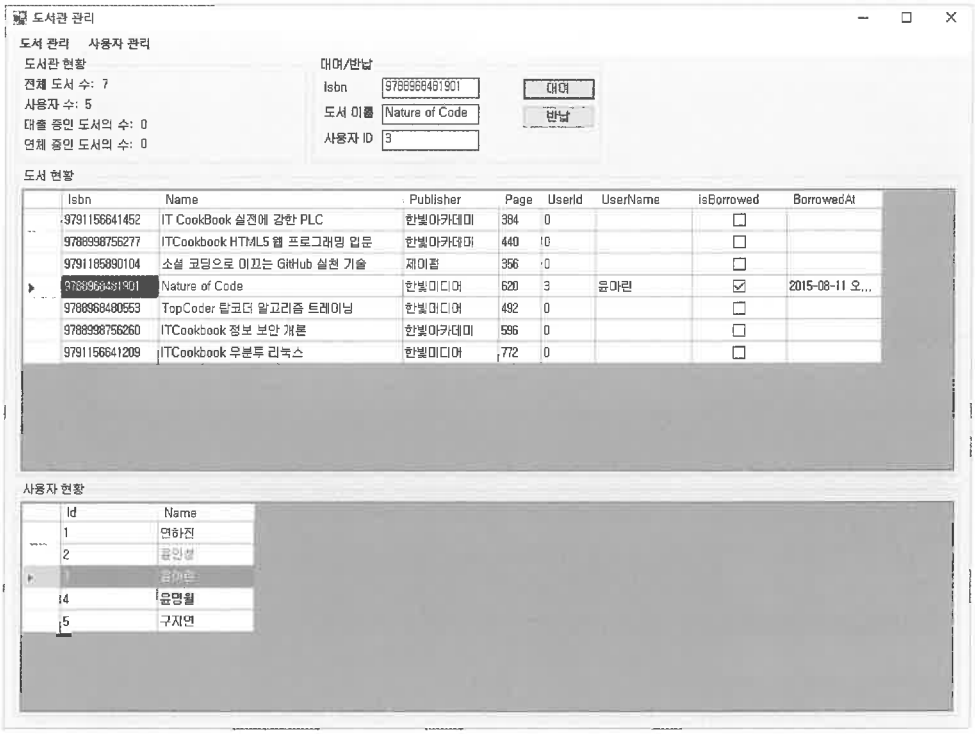

그림 13-3 대여 상태가 그리드에 출력

대여된 도서를 클릭하고 [반납] 버튼을 누르면 반납과 관련된 메시지 상자가 출력됩니다.

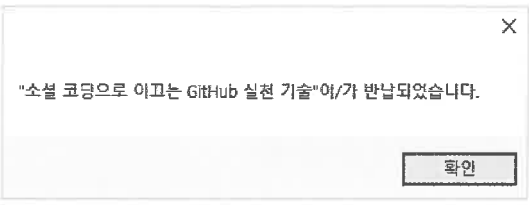

그림 13-4 반납 관련 메시지 상자

이어서 첫 번째 메뉴, 도서 관리입니다. 도서 관리 화면에서는 도서를 추가/수정/삭제가 가능합니다. 도서 추가의 경우는 정보를 입력하고 [추가] 버튼을 누르고, 수정의 경우는 도서 현황에서 도서를 선택하고 정보를 수정한 이후에 [수정] 버튼을 누르고, 삭제의 경우는 도서 현황에서 도서를 선택한 뒤에 [삭제] 버튼을 누르면 됩니다.

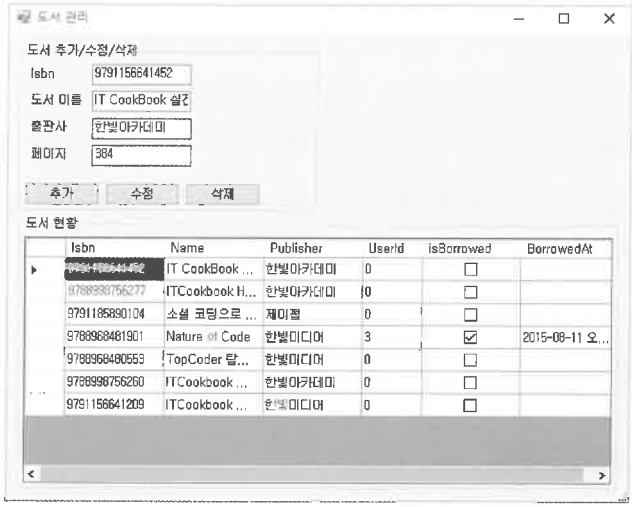

그림 13-5 도서 관리 메뉴

두 번째 메뉴는 사용자 관리입니다. 사용자 관리 화면에서는 이전과 마찬가지로 사용자 정보를 추가/수정/삭제할 수 있습니다.

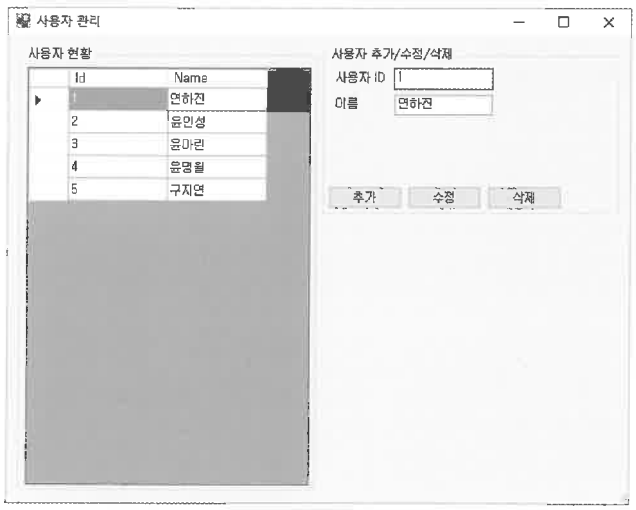

그림 13-6 사용자 관리 메뉴

02 프로젝트 구성

프로젝트의 기본 구성은 [그림 13-7]과 같습니다. 기본적으로 윈도 폼 프로젝트를 만들면서 생성되는 Form1.cs 파일 이외에 Form2.cs, Form3.cs라는 이름으로 윈도 폼을 추가했습니다. 또한 데이터 관리를 위해 DataManager.cs, Book.cs, User.cs 클래스를 추가했습니다.

데이터는 데이터베이스를 배우지 않았으므로, 데이터베이스를 사용하지 않고 XML을 사용해 저장하고 불러옵니다. 그럼 지금부터 차근차근 코드를 입력해보겠습니다.

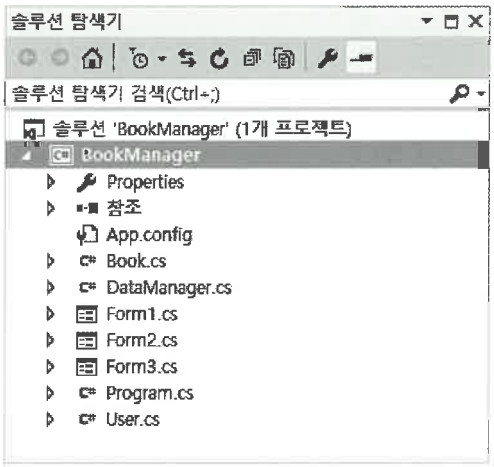

그림 13-7 프로젝트 구성

03 모델 클래스

일단 데이터를 나타내는 모델 클래스부터 만들겠습니다. 현재 예제에서는 두 개의 모델을 활용하고 있는데요. 도서와 사용자입니다.

도서를 나타내는 Book 클래스는 다음과 같이 구성합니다.

코드 13-1 Book 클래스

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

namespace BookManager
{
    class Book
    {
        public string Isbn { get; set; }
        public string Name { get; set; }
        public string Publisher { get; set; }
        public int Page { get; set; }

        public int UserId { get; set; }
        public string UserName { get; set; }
        public bool isBorrowed { get; set; }
        public DateTime BorrowedAt { get; set; }
    }
}
```

사용자를 나타내는 User 클래스는 다음과 같이 구성합니다.

코드 13-2 User 클래스

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

namespace BookManager
{
    class User
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

04 DataManager 클래스

데이터베이스를 배우지 않았으므로 데이터베이스 대신 XML을 사용해 데이터를 관리합니다. 이후에 데이터베이스를 배운다면 이 코드를 데이터베이스가 알아서 처리해준답니다.

코드 13-3 DataManager 클래스

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

namespace BookManager
{
    class DataManager
    {
        public static List<Book> Books = new List<Book>();
        public static List<User> Users = new List<User>();

        static DataManager()
        {
            Load();
        }
```

모든 클래스에서 접근할 수 있게 static으로 데이터 리스트를 만들어 줍니다. 이렇게 데이터를 모든 클래스에서 접근할 수 있게 만든 리스트를 Static 저장소(Storage) 기술이라도 부릅니다.

```
public static void Load()
{
   try
   {
      string booksOutput = File.ReadAllText(@"./Books.xml");
      XElement booksXElement = XElement.Parse(booksOutput);
      Books = (from item in booksXElement.Descendants("book")
               select new Book()
              {
                 Isbn = item.Element("isbn").Value,
                 Name = item.Element("name").Value,
                 Publisher = item.Element("publisher").Value,
                 Page = int.Parse(item.Element("page").Value),
                 BorrowedAt = DateTime.Parse(item.Element("borrowedAt").Value),
                 isBorrowed = item.Element("isBorrowed").Value != "0" ? true : false,
                 UserId = int.Parse(item.Element("userId").Value),
                 UserName = item.Element("userName").Value
              }).ToList<Book>();

      string usersOutput = File.ReadAllText(@"./Users.xml");
      XElement usersXElement = XElement.Parse(usersOutput);
      Users = (from item in usersXElement.Descendants("user")
               select new User()
               {
                 Id = int.Parse(item.Element("id").Value),
                 Name = item.Element("name").Value
              }).ToList<User>();
   }
   catch (FileNotFoundException exception)
   {
      Save();
   }
}
```

도서 XML을 읽고 파싱합니다.

삼항 조건부 연산자를 사용했습니다.

사용자 XML을 읽고 파싱합니다.

처음 프로젝트를 실행하면 Books.xml 파일과 Users.xml 파일이 존재하지 않습니다. 이러한 상황에는 FileLoadException이 발생하는데요. 이때는 Save() 메서드를 호출해서 빈 XML 파일을 2개 만들어줍니다.

```
        public static void Save()
        {
            string booksOutput = "";
            booksOutput += "<books>\n";
            foreach (var item in Books)
            {
                booksOutput += "<book>\n";
                booksOutput += "  <isbn>" + item.Isbn + "</isbn>\n";
                booksOutput += "  <name>" + item.Name + "</name>\n";
                booksOutput += "  <publisher>" + item.Publisher + "</publisher>\n";
                booksOutput += "  <page>" + item.Page + "</page>\n";
                booksOutput += "  <borrowedAt>" + item.BorrowedAt.ToLongDateString()
                                                + "</borrowedAt>\n";
                booksOutput += "  <isBorrowed>" + (item.isBorrowed ? 1 : 0)
                                                + "</isBorrowed>\n";
                booksOutput += "  <userId>" + item.UserId + "</userId>\n";
                booksOutput += "  <userName>" + item.UserName + "</userName>\n";
                booksOutput += "</book>\n";
            }
            booksOutput += "</books>";

            string usersOutput = "";
            usersOutput += "<users>\n";
            foreach (var item in Users)
            {
                usersOutput += "<user>\n";
                usersOutput += "  <id>" + item.Id + "</id>\n";
                usersOutput += "  <name>" + item.Name + "</name>\n";
                usersOutput += "</user>\n";
            }
            usersOutput += "</users>";

            File.WriteAllText(@"./Books.xml", booksOutput);
            File.WriteAllText(@"./Users.xml", usersOutput);
        }
    }
}
```

도서 정보를 XML로 만듭니다.

삼항 조건부 연산자를 사용했습니다.

사용자 정보를 XML로 만듭니다.

저장합니다.

이러한 DataManager.cs 파일을 사용하면, 다음과 같은 형태의 XML 파일을 읽고 저장하게 됩니다.

코드 13-4 **DataManager 클래스가 관리하는 XML 파일의 형태**

```
<books>
  <book>
    <isbn>9791156641452</isbn>
    <name>IT CookBook 실전에 강한 PLC</name>
    <publisher>한빛아카데미</publisher>
    <page>384</page>
    <borrowedAt>0001년 1월 1일 월요일</borrowedAt>
    <isBorrowed>0</isBorrowed>
    <userId>0</userId>
    <userName></userName>
  </book>
  <book>
    <isbn>9788998756277</isbn>
    <name>ITCookbook HTML5 웹 프로그래밍 입문</name>
    <publisher>한빛아카데미</publisher>
    <page>440</page>
    <borrowedAt>0001년 1월 1일 월요일</borrowedAt>
    <isBorrowed>0</isBorrowed>
    <userId>0</userId>
    <userName></userName>
  </book>
</books>
<users>
  <user>
    <id>1</id>
    <name>연하진</name>
  </user>
  <user>
    <id>2</id>
    <name>윤인성</name>
  </user>
</users>
```

05 디자인

데이터를 모두 구성했으니, 이제 화면을 디자인하고 속성 지정과 이벤트 연결만 해주면 끝입니다. 일단 Form1.cs 파일을 디자인에서 다음과 같이 디자인해줍니다.

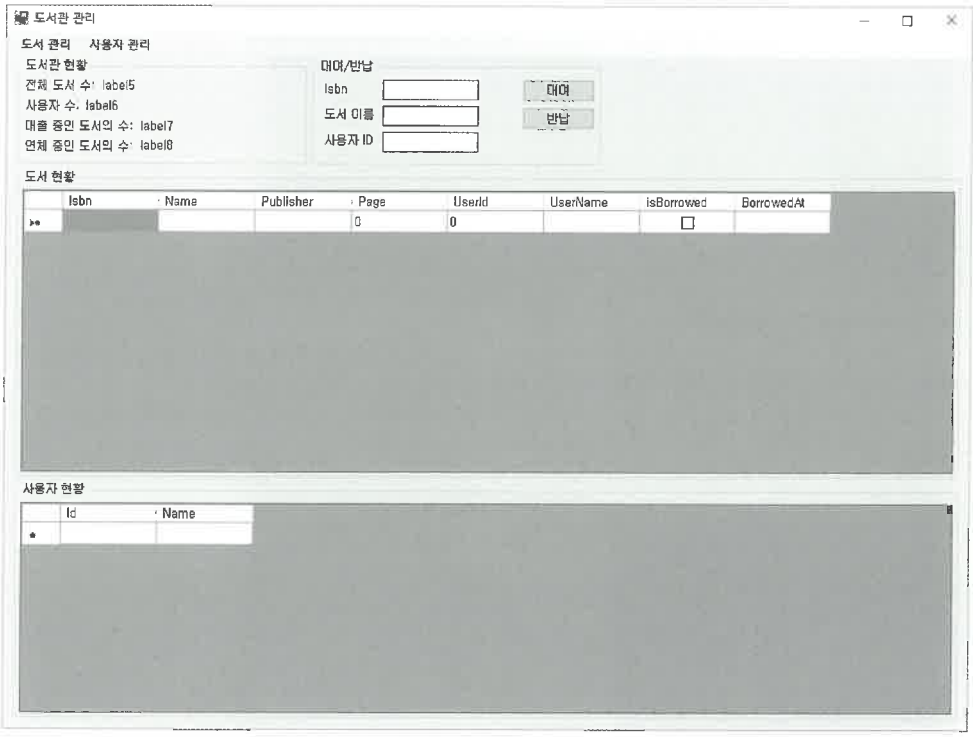

그림 13-8 Form1.cs 파일 디자인

Form2.cs 파일도 디자인에서 다음과 같이 디자인해줍니다.

그림 13-9 Form2.cs 파일 디자인

Form3.cs 파일은 다음과 같이 디자인해줍니다.

그림 13-10 Form3.cs 파일 디자인

06 데이터 소스 구성

화면을 디자인하면서 데이터 그리드를 사용했는데요, 이곳에 데이터 개체도 지정해줍니다. 데이터 개체는 클래스를 기반으로, 이전에 만들었던 Book 클래스와 User 클래스를 지정해줍니다.

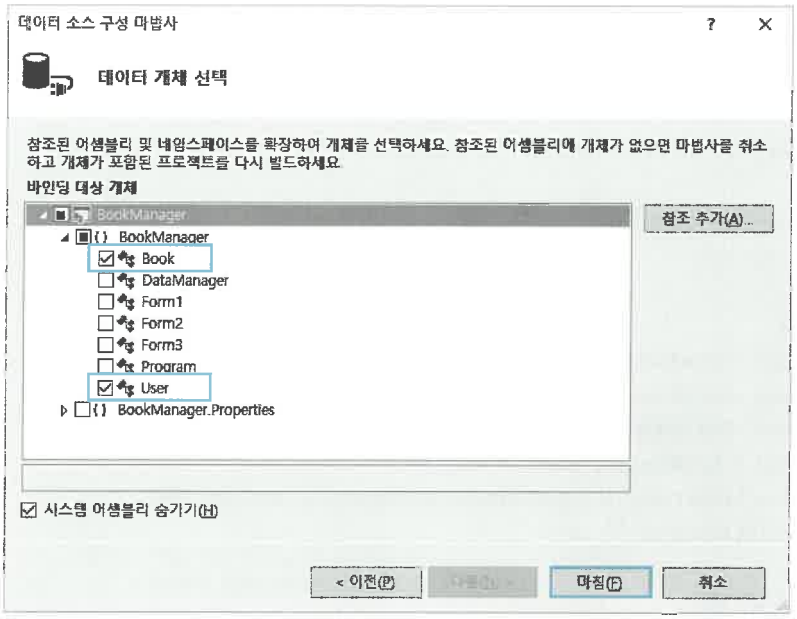

그림 13-11 데이터 소스 구성

이렇게 생성한 데이터 소스를 디자인 화면에서 데이터 그리드와 연결하기 바랍니다.

07 속성 지정과 이벤트 연결

마지막으로 윈도 폼 코드를 작성하겠습니다. 이벤트 연결 등을 모두 쉽게 확인할 수 있게 코드로 지정했습니다. 또한 책의 내용을 전체적으로 활용하기 위해 이벤트를 연결할 때 메서드 이름과 람다를 모두 활용해보았습니다.

코드 13-5 Form1.cs 파일

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        Text = "도서관 관리";

        // 라벨 설정
        label5.Text = DataManager.Books.Count.ToString();
        label6.Text = DataManager.Users.Count.ToString();
        label7.Text = DataManager.Books.Where((x) => x.isBorrowed).Count().ToString();
        label8.Text = DataManager.Books.Where((x) => {
            return x.isBorrowed && x.BorrowedAt.AddDays(7) < DateTime.Now;
        }).Count().ToString();

        // 데이터 그리드 설정
        dataGridView1.DataSource = DataManager.Books;
        dataGridView2.DataSource = DataManager.Users;
        dataGridView1.CurrentCellChanged += DataGridView1_CurrentCellChanged;
        dataGridView2.CurrentCellChanged += DataGridView2_CurrentCellChanged;

        // 버튼 이벤트 설정
        button1.Click += Button1_Click;
        button2.Click += Button2_Click;
    }

    private void DataGridView1_CurrentCellChanged(object sender, EventArgs e)
```

Where() 메서드는 리스트에서 조건에 맞는 대상을 뽑아내는 메서드입니다. isBorrowed 속성이 true인 객체들만 리스트로 뽑아 크기를 구합니다.

대여 기간은 7일입니다. "빌린 날짜 + 7일"보다 현재 시간이 뒤라면 연체된 것이겠죠?

```
{
    try
    {
        // 그리드의 셀이 선택되면 텍스트 박스에 글자 지정
        Book book = dataGridView1.CurrentRow.DataBoundItem as Book;
        textBox1.Text = book.Isbn;
        textBox2.Text = book.Name;
    }
    catch (Exception exception)
    {

    }
}

private void DataGridView2_CurrentCellChanged(object sender, EventArgs e)
{
    try
    {
        // 그리드의 셀이 선택되면 텍스트 박스에 글자 지정
        User book = dataGridView2.CurrentRow.DataBoundItem as User;
        textBox3.Text = book.Id.ToString();
    }
    catch (Exception exception)
    {

    }
}

private void Button1_Click(object sender, EventArgs e)
{
    if (textBox1.Text.Trim() == "")
    {
        MessageBox.Show("Isbn을 입력해주세요");
    }
    else if (textBox3.Text.Trim() == "")
    {
        MessageBox.Show("사용자 Id를 입력해주세요");
    }
    else
    {
        try
        {
```

초기에 셀이 선택되지 않은 상태에서는 예외가 발생할 수 있으므로 예외 처리를 했습니다.

```
            Book book = DataManager.Books.Single((x) => x.Isbn == textBox1.Text);
            if (book.isBorrowed)
            {
                MessageBox.Show("이미 대여 중인 도서입니다.");
            }
            else
            {
                User user = DataManager.Users.Single((x) => x.Id.ToString() == textBox3.Text);
                book.UserId = user.Id;
                book.UserName = user.Name;
                book.isBorrowed = true;
                book.BorrowedAt = DateTime.Now;

                dataGridView1.DataSource = null;
                dataGridView1.DataSource = DataManager.Books;
                DataManager.Save();

                MessageBox.Show("\"" + book.Name + "\"이/가 \"" + user.Name + "\"님께 대여되었습니다.");
            }
        }
        catch (Exception exception)
        {
            MessageBox.Show("존재하지 않는 도서 또는 사용자입니다.");
        }
    }
}

private void Button2_Click(object sender, EventArgs e)
{
    if (textBox1.Text.Trim() == "")
    {
        MessageBox.Show("Isbn을 입력해주세요");
    }
    else
    {
        try
        {
            Book book = DataManager.Books.Single((x) => x.Isbn == textBox1.Text);
            if (book.isBorrowed)
            {
                User user = DataManager.Users.Single((x) => x.Id.ToString() == book.
                                                        UserId.ToString());
```

Single() 메서드는 조건에 맞는 대상 객체 하나를 추출하는 메서드입니다. 만약 조건에 맞는 대상이 없다면 예외가 발생합니다.

그리드를 새로고침하고 XML로 저장하는 코드입니다. 굉장히 자주 반복해서 사용하는 코드입니다.

조건에 맞는 대상이 없다면 존재하지 않는 도서 또는 사용자입니다라는 문자열을 출력합니다.

```
                book.UserId = 0;
                book.UserName = "";
                book.isBorrowed = false;
                book.BorrowedAt = new DateTime();

                dataGridView1.DataSource = null;
                dataGridView1.DataSource = DataManager.Books;
                DataManager.Save();

                if (book.BorrowedAt.AddDays(7) > DateTime.Now)
                {
                    MessageBox.Show("\"" + book.Name + "\"이/가 연체 상태로 반납되었습니다.");
                }
                else
                {
                    MessageBox.Show("\"" + book.Name + "\"이/가 반납되었습니다.");
                }
            }
            else
            {
                MessageBox.Show("대여 상태가 아닙니다.");
            }
        }
        catch (Exception exception)
        {
            MessageBox.Show("존재하지 않는 도서 또는 사용자입니다.");
        }
    }
}

private void 도서관리ToolStripMenuItem_Click(object sender, EventArgs e)
{
    new Form2().ShowDialog();
}

private void 사용자관리ToolStripMenuItem_Click(object sender, EventArgs e)
{
    new Form3().ShowDialog();
}
}
```

메뉴를 누르면 새로운 윈도 폼을 출력합니다.

코드 13-6 Form2.cs 파일

```
public partial class Form2 : Form
{
    public Form2()
    {
        InitializeComponent();
        Text = "도서 관리";

        // 데이터 그리드 설정
        dataGridView1.DataSource = DataManager.Books;
        dataGridView1.CurrentCellChanged += DataGridView1_CurrentCellChanged;

        // 버튼 설정
        button1.Click += (sender, e) =>
        {
            // 추가 버튼
            try
            {
                if (DataManager.Books.Exists((x) => x.Isbn == textBox1.Text))
                {
                    MessageBox.Show("이미 존재하는 도서입니다");
                }
                else
                {
                    Book book = new Book()
                    {
                        Isbn = textBox1.Text,
                        Name = textBox2.Text,
                        Publisher = textBox3.Text,
                        Page = int.Parse(textBox4.Text)
                    };
                    DataManager.Books.Add(book);

                    dataGridView1.DataSource = null;
                    dataGridView1.DataSource = DataManager.Books;
                    DataManager.Save();
                }
            }
            catch (Exception exception)
            {
```

Exists() 메서드는 리스트에 조건에 맞는 객체가 있는지 확인하는 메서드입니다. 이전에는 고급 예외 처리를 사용해 조건을 분기했는데요. 여기서는 기본 예외 처리를 사용했다고 볼 수 있지요.

```
        }
    };

    button2.Click += (sender, e) =>
    {
        // 수정 버튼
        try
        {
            Book book = DataManager.Books.Single((x) => x.Isbn == textBox1.Text);
            book.Name = textBox2.Text;
            book.Publisher = textBox3.Text;
            book.Page = int.Parse(textBox4.Text);

            dataGridView1.DataSource = null;
            dataGridView1.DataSource = DataManager.Books;
            DataManager.Save();
        }
        catch (Exception exception)
        {
            MessageBox.Show("존재하지 않는 도서입니다");
        }
    };

    button3.Click += (sender, e) =>
    {
        // 수정 버튼
        try
        {
            Book book = DataManager.Books.Single((x) => x.Isbn == textBox1.Text);
            DataManager.Books.Remove(book);

            dataGridView1.DataSource = null;
            dataGridView1.DataSource = DataManager.Books;
            DataManager.Save();
        }
        catch (Exception exception)
        {
            MessageBox.Show("존재하지 않는 도서입니다");
        }
    };
}
```

```
    private void DataGridView1_CurrentCellChanged(object sender, EventArgs e)
    {
        try
        {
            // 그리드의 셀이 선택되면 텍스트 박스에 글자 지정
            Book book = dataGridView1.CurrentRow.DataBoundItem as Book;
            textBox1.Text = book.Isbn;
            textBox2.Text = book.Name;
            textBox3.Text = book.Publisher;
            textBox4.Text = book.Page.ToString();
        }
        catch (Exception exception)
        {

        }
    }
}
```

코드 13-7 **Form3.cs 파일**

```
public partial class Form3 : Form
{
    public Form3()
    {
        InitializeComponent();
        Text = "사용자 관리";

        // 데이터 그리드 설정
        dataGridView1.DataSource = DataManager.Users;
        dataGridView1.CurrentCellChanged += DataGridView1_CurrentCellChanged;

        // 버튼 설정
        button1.Click += (sender, e) =>
        {
            // 추가 버튼
            try
            {
                if (DataManager.Users.Exists((x) => x.Id == int.Parse(textBox1.Text)))
                {
                    MessageBox.Show("사용자 ID가 겹칩니다");
                }
```

```
        else
        {
            User user = new User()
            {
                Id = int.Parse(textBox1.Text),
                Name = textBox2.Text
            };
            DataManager.Users.Add(user);

            dataGridView1.DataSource = null;
            dataGridView1.DataSource = DataManager.Users;
            DataManager.Save();
        }
    }
    catch (Exception exception)
    {

    }
};

button2.Click += (sender, e) =>
{
    // 수정 버튼
    try
    {
        User user = DataManager.Users.Single((x) => x.Id == int.Parse(textBox1.Text));
        user.Name = textBox2.Text;

        dataGridView1.DataSource = null;
        dataGridView1.DataSource = DataManager.Users;
    }
    catch (Exception exception)
    {
        MessageBox.Show("존재하지 않는 사용자입니다");
    }
};
```

```
        button3.Click += (sender, e) =>
        {
            // 수정 버튼
            try
            {
                User user = DataManager.Users.Single((x) => x.Id == int.Parse(textBox1.Text));
                DataManager.Users.Remove(user);

                dataGridView1.DataSource = null;
                dataGridView1.DataSource = DataManager.Users;
                DataManager.Save();
            }
            catch (Exception exception)
            {
                MessageBox.Show("존재하지 않는 사용자입니다");
            }
        };
    }

    private void DataGridView1_CurrentCellChanged(object sender, EventArgs e)
    {
        try
        {
            // 그리드의 셀이 선택되면 텍스트 박스에 글자 지정
            User user = dataGridView1.CurrentRow.DataBoundItem as User;
            textBox1.Text = user.Id.ToString();
            textBox2.Text = user.Name;
        }
        catch (Exception exception)
        {

        }
    }
}
```

코드를 읽고 잘 기억나지 않는 부분이 있다면, 관련된 부분을 다시 살펴보기 바랍니다.

08 책의 내용을 마치며

책의 내용을 모두 마쳤습니다. C#과 관련된 기본 내용을 대부분 다루었지만 C#은 이 외에도 더 많은 내용이 있습니다. 이 책을 다 공부하고 방학 동안에 다음과 같은 프레임워크 또는 엔진을 스스로 공부해보면 도움이 될 것이라 생각합니다. 자신이 만들고 싶은 것에 맞춰 공부하면 더 재미있겠죠?

- **웹 개발**: ASP.NET, ASP.NET MVC
- **응용 프로그램 개발**: WPF, 윈도 10 유니버셜 응용 프로그램
- **IoT 개발**: 윈도 10 라즈베리 파이
- **게임 개발**: 유니티

C#을 사용한 프레임워크들은 마이크로소프트 사라는 거대 기업 아래에서 굉장히 잘 만들어져 있습니다. 내부에서 사용되는 디자인 패턴 등을 이해하면서 공부하면, 이후에 다른 언어와 응용 프로그램을 사용해 무언가를 만들 때에도 큰 도움이 될 것입니다.

기호

A~C

D~J

K~O