```
// Program 2
// CIS 200-01
// Due: 4/5/2018
// Grading ID: Z5860

// File: Prog3Form.cs
// This class creates the main GUI for Program 3. It provides a
// File menu with About, Exit, Open, and Save items, an Insert menu with Patron and
// Book items, an Item menu with Check Out and Return items, an Edit menu, and a
// Report menu with Patron List, Item List, and Checked Out Items items.


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using System.Runtime.Serialization;

namespace LibraryItems
{
   public partial class Prog3Form : Form
   {
      private Library _lib; // The library
      private BinaryFormatter formatter = new BinaryFormatter();
      private BinaryFormatter reader = new BinaryFormatter();
      private FileStream output;
      private FileStream input;


      // Precondition:  None
      // Postcondition: The form's GUI is prepared for display. A library object.
      public Prog3Form()
      {
         InitializeComponent();

         _lib = new Library(); // Create the library

      }
```

```
// Precondition:  File, About menu item activated
// Postcondition: Information about author displayed in dialog box
private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
{
   string NL = Environment.NewLine; // NewLine shortcut

   MessageBox.Show($"Program 3{NL}Grading ID: Z5860{NL}CIS 200-01{NL}April 5th, 2018",
      "About Program 3");
}

// Precondition:  File, Exit menu item activated
// Postcondition: The application is exited
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
   Application.Exit();
}

// Precondition:  Report, Patron List menu item activated
// Postcondition: The list of patrons is displayed in the reportTxt
//                text box
private void patronListToolStripMenuItem_Click(object sender, EventArgs e)
{
   StringBuilder result = new StringBuilder(); // Holds text as report being built
                                    // StringBuilder more efficient than String
   List<LibraryPatron> patrons;              // List of patrons
   string NL = Environment.NewLine;          // NewLine shortcut

   patrons = _lib.GetPatronsList();

   result.Append($"Patron List - {patrons.Count} patrons{NL}{NL}");

   foreach (LibraryPatron p in patrons)
      result.Append($"{p}{NL}{NL}");

   reportTxt.Text = result.ToString();

   // Put cursor at start of report
   reportTxt.SelectionStart = 0;
}

// Precondition:  Report, Item List menu item activated
// Postcondition: The list of items is displayed in the reportTxt
//                text box
```

```
private void itemListToolStripMenuItem_Click(object sender, EventArgs e)
{
   StringBuilder result = new StringBuilder(); // Holds text as report being built
                                  // StringBuilder more efficient than String
   List<LibraryItem> items;                // List of library items
   string NL = Environment.NewLine;          // NewLine shortcut

   items = _lib.GetItemsList();

   result.Append($"Item List - {items.Count} items{NL}{NL}");

   foreach (LibraryItem item in items)
      result.Append($"{item}{NL}{NL}");

   reportTxt.Text = result.ToString();

   // Put cursor at start of report
   reportTxt.SelectionStart = 0;
}

// Precondition:  Report, Checked Out Items menu item activated
// Postcondition: The list of checked out items is displayed in the
//             reportTxt text box
private void checkedOutItemsToolStripMenuItem_Click(object sender, EventArgs e)
{
   StringBuilder result = new StringBuilder(); // Holds text as report being built
                                  // StringBuilder more efficient than String
   List<LibraryItem> items;                // List of library items
   string NL = Environment.NewLine;          // NewLine shortcut

   items = _lib.GetItemsList();

   // LINQ: selects checked out items
   var checkedOutItems =
      from item in items
      where item.IsCheckedOut()
      select item;

   result.Append($"Checked Out Items - {checkedOutItems.Count()} items{NL}{NL}");

   foreach (LibraryItem item in checkedOutItems)
      result.Append($"{item}{NL}{NL}");

   reportTxt.Text = result.ToString();
```

```
        // Put cursor at start of report
        reportTxt.SelectionStart = 0;
    }

    // Precondition:  Insert, Patron menu item activated
    // Postcondition: The Patron dialog box is displayed. If data entered
    //              are OK, a LibraryPatron is created and added to the library
    private void patronToolStripMenuItem_Click(object sender, EventArgs e)
    {
        PatronForm patronForm = new PatronForm(); // The patron dialog box form

        DialogResult result = patronForm.ShowDialog(); // Show form as dialog and store result

        if (result == DialogResult.OK) // Only add if OK
        {
            // Use form's properties to get patron info to send to library
            _lib.AddPatron(patronForm.PatronName, patronForm.PatronID);
        }

        patronForm.Dispose(); // Good .NET practice - will get garbage collected anyway
    }

    // Precondition:  Insert, Book menu item activated
    // Postcondition: The Book dialog box is displayed. If data entered
    //              are OK, a LibraryBook is created and added to the library
    private void bookToolStripMenuItem_Click(object sender, EventArgs e)
    {
        BookForm bookForm = new BookForm(); // The book dialog box form

        DialogResult result = bookForm.ShowDialog(); // Show form as dialog and store result

        if (result == DialogResult.OK) // Only add if OK
        {
            try
            {
                // Use form's properties to get book info to send to library
                _lib.AddLibraryBook(bookForm.ItemTitle, bookForm.ItemPublisher,
int.Parse(bookForm.ItemCopyrightYear),
                    int.Parse(bookForm.ItemLoanPeriod), bookForm.ItemCallNumber, bookForm.BookAuthor);
            }

            catch (FormatException) // This should never happen if form validation works!
            {
```

```
            MessageBox.Show("Problem with Book Validation!", "Validation Error");
          }
        }

        bookForm.Dispose(); // Good .NET practice - will get garbage collected anyway
      }

      // Precondition:  Item, Check Out menu item activated
      // Postcondition: The Checkout dialog box is displayed. If data entered
      //             are OK, an item is checked out from the library by a patron
      private void checkOutToolStripMenuItem_Click(object sender, EventArgs e)
      {
        List<LibraryItem> items;     // List of library items
        List<LibraryPatron> patrons; // List of patrons

        items = _lib.GetItemsList();
        patrons = _lib.GetPatronsList();

        if (((items.Count - _lib.GetCheckedOutCount()) == 0) || (patrons.Count() == 0)) // Must have
items and patrons
          MessageBox.Show("Must have items and patrons to check out!", "Check Out Error");
        else
        {
          CheckoutForm checkoutForm = new CheckoutForm(items, patrons); // The check out dialog box
form

          DialogResult result = checkoutForm.ShowDialog(); // Show form as dialog and store result

          if (result == DialogResult.OK) // Only add if OK
          {
            _lib.CheckOut(checkoutForm.ItemIndex, checkoutForm.PatronIndex);
          }

          checkoutForm.Dispose(); // Good .NET practice - will get garbage collected anyway
        }
      }

      // Precondition:  Item, Return menu item activated
      // Postcondition: The Return dialog box is displayed. If data entered
      //             are OK, an item is returned to the library
      private void returnToolStripMenuItem_Click(object sender, EventArgs e)
      {
        List<LibraryItem> items;     // List of library items
```

```
        items = _lib.GetItemsList();

        if ((_lib.GetCheckedOutCount() == 0) ) // Must have items to return
            MessageBox.Show("Must have items to return!", "Return Error");
        else
        {
            ReturnForm returnForm = new ReturnForm(items); // The return dialog box form

            DialogResult result = returnForm.ShowDialog(); // Show form as dialog and store result

            if (result == DialogResult.OK) // Only add if OK
            {
                _lib.ReturnToShelf(returnForm.ItemIndex);
            }

            returnForm.Dispose(); // Good .NET practice - will get garbage collected anyway
        }
    }

    // Precondition:  File, Save menu item activated
    // Postcondition: The SaveFileDialog box is displayed. If data entered
    //              are OK, the library is saved to the file
    private void saveToolStripMenuItem_Click(object sender, EventArgs e)
    {

        DialogResult result;  // OK or cancel
        string fileName;  // name of file saved

        using (SaveFileDialog fileChooser = new SaveFileDialog())
        {
            fileChooser.CheckFileExists = false;

            result = fileChooser.ShowDialog();
            fileName = fileChooser.FileName;
        }

        if (result == DialogResult.OK)
        {
            if (string.IsNullOrEmpty(fileName))
            {
                MessageBox.Show("Invalad File Name", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            else
```

```
            {
               try
               {
                  output = new FileStream(fileName, FileMode.Create, FileAccess.Write);
                  formatter.Serialize(output, _lib);
               }
               catch (IOException)
               {
                  MessageBox.Show("Error Saving File", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
               }
            }
         }
      }

      // Precondition:  File, Open menu item activated
      // Postcondition: The OpenFileDialog box is displayed. If data entered
      //              are OK, the library is open to the application
      private void openToolStripMenuItem_Click(object sender, EventArgs e)
      {
         DialogResult result;  //OK or cancel
         string fileName;    // name of the file opened

         using (OpenFileDialog fileChooser = new OpenFileDialog())
         {
            result = fileChooser.ShowDialog();
            fileName = fileChooser.FileName;
         }

         if (result == DialogResult.OK)
         {
            if(string.IsNullOrEmpty(fileName))
            {
               MessageBox.Show("Invalid File Name", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            else
            {
               input = new FileStream(fileName, FileMode.Open, FileAccess.Read);
               try
               {
                  _lib = (Library)reader.Deserialize(input);
               }
               catch (SerializationException)
```

```
            {
               input?.Close();
            }
         }
      }
   }

   // Precondition:  Edit menu item activated
   // Postcondition: The LibraryPatron box is displayed. If data entered
   //              are OK, the library is saved to the file
   private void editPatronToolStripMenuItem_Click(object sender, EventArgs e)
   {
      List<LibraryPatron> patrons; // List of patrons

      patrons = _lib.GetPatronsList();

      if (patrons.Count() == 0) // Must have patrons
         MessageBox.Show("Must have items and patrons to check out!", "Check Out Error");
      else
      {
         ChoosePatronForm choosePatronForm = new ChoosePatronForm(patrons); // The Choose
Patron dialog box form

         DialogResult result = choosePatronForm.ShowDialog(); // Show form as dialog and store result

         if (result == DialogResult.OK) // Only add if OK
         {
            int pIndex = choosePatronForm.PatronIndex;
            LibraryPatron patron = patrons[pIndex];

            PatronForm editPatronForm = new PatronForm(); // The patron form dialog box form
            editPatronForm.PatronName = patron.PatronName;
            editPatronForm.PatronID = patron.PatronID;

            DialogResult resultEditForm = editPatronForm.ShowDialog();

            if(resultEditForm == DialogResult.OK) // Edit only if OK
            {
               patron.PatronName = editPatronForm.PatronName;
               patron.PatronID = editPatronForm.PatronID;
            }

            editPatronForm.Dispose();
         }
```

```
            choosePatronForm.Dispose(); // Good .NET practice - will get garbage collected anyway
        }
    }

    private void editBookToolStripMenuItem_Click(object sender, EventArgs e)
    {
        List<LibraryItem> items; // List of items

        items = _lib.GetItemsList();

        if (items.Count() == 0) // Must have items
            MessageBox.Show("Must have items to make edits!", "Check Out Error");
        else
        {
            ChooseBookForm chooseBookForm = new ChooseBookForm(items); // The Choose Book dialog
box form

            DialogResult result = chooseBookForm.ShowDialog(); // Show form as dialog and store result

            if (result == DialogResult.OK) // Only add if OK
            {
                int bIndex = chooseBookForm.ItemIndex;
                LibraryBook book = (LibraryBook)items[bIndex];

                BookForm editBookForm = new BookForm(); // The LibraryBook dialog box form

                editBookForm.ItemTitle = book.Title;
                editBookForm.ItemPublisher = book.Publisher;
                editBookForm.ItemCopyrightYear = Convert.ToString(book.CopyrightYear);
                editBookForm.ItemLoanPeriod = Convert.ToString(book.LoanPeriod);
                editBookForm.ItemCallNumber = book.CallNumber;
                editBookForm.BookAuthor = book.Author;

                DialogResult resultEditForm = editBookForm.ShowDialog();

                if (resultEditForm == DialogResult.OK)
                {
                    book.Title = editBookForm.ItemTitle;
                    book.Publisher = editBookForm.ItemPublisher;
                    book.CopyrightYear = int.Parse(editBookForm.ItemCopyrightYear);
                    book.LoanPeriod = int.Parse(editBookForm.ItemLoanPeriod);
                    book.CallNumber = editBookForm.ItemCallNumber;
                    book.Author = editBookForm.BookAuthor;
```

```
            }

            editBookForm.Dispose();
          }

          chooseBookForm.Dispose(); // Good .NET practice - will get garbage collected anyway
        }
      }
    }
}
```

```
// Program 2
// CIS 200-01
// Spring 2018
// By: Andrew L. Wright

// File: Library.cs
// This file creates a basic Library class that stores a list
// of LibraryItems and a list of LibraryPatrons. It allows items
// to be checked out by patrons. The lists are accessible to other
// classes in the same namespace (LibraryItems).

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LibraryItems
{
    [Serializable]
    public class Library
    {
        // Namespace Accessible Data - Use with care
        internal List<LibraryItem> _items;     // List of items stored in Library
        internal List<LibraryPatron> _patrons; // List of patrons of Library

        // Precondition:  None
        // Postcondition: The library has been created and is empty (no books, no
patrons)
        public Library()
        {
            _items = new List<LibraryItem>();
            _patrons = new List<LibraryPatron>();
        }

        // Precondition:  None
        // Postcondition: A patron has been created with the specified values for name
and ID.
        //                The patron has been added to the Library.
        public void AddPatron(String name, String id)
        {
            _patrons.Add(new LibraryPatron(name, id));
        }

        // Precondition:  theCopyrightYear >= 0 and theLoanPeriod >= 0
        // Postcondition: A library book has been created with the specified
        //                values for title, publisher, copyright year, loan period,
        //                call number, and author. The item is not checked out.
        //                The book has been added to the Library.
        public void AddLibraryBook(String theTitle, String thePublisher, int
theCopyrightYear,
            int theLoanPeriod, String theCallNumber, String theAuthor)
        {
            _items.Add(new LibraryBook(theTitle, thePublisher, theCopyrightYear,
theLoanPeriod,
                theCallNumber, theAuthor));
        }

        // Precondition:  theCopyrightYear >= 0 and theLoanPeriod >= 0 and
```

```
        //                theMedium from { DVD, BLURAY, VHS } and theDuration >= 0
        // Postcondition: A library movie has been created with the specified
        //                values for title, publisher, copyright year, loan period,
        //                call number, duration, director, medium, and rating. The
        //                item is not checked out.
        //                The movie has been added to the Library.
        public void AddLibraryMovie(String theTitle, String thePublisher, int
theCopyrightYear,
            int theLoanPeriod, String theCallNumber, double theDuration, String
theDirector,
            LibraryMediaItem.MediaType theMedium, LibraryMovie.MPAARatings theRating)
        {
            _items.Add(new LibraryMovie(theTitle, thePublisher, theCopyrightYear,
theLoanPeriod,
                theCallNumber, theDuration, theDirector, theMedium, theRating));
        }

        // Precondition:  theCopyrightYear >= 0 and theLoanPeriod >= 0 and
        //                theMedium from { CD, SACD, VINYL } and theDuration >= 0 and
        //                theNumTracks >= 0
        // Postcondition: A library music item has been created with the specified
        //                values for title, publisher, copyright year, loan period,
        //                call number, duration, director, medium, and rating. The
        //                item is not checked out.
        //                The music item has been added to the Library.
        public void AddLibraryMusic(String theTitle, String thePublisher, int
theCopyrightYear,
            int theLoanPeriod, String theCallNumber, double theDuration, String
theArtist,
            LibraryMediaItem.MediaType theMedium, int theNumTracks)
        {
            _items.Add(new LibraryMusic(theTitle, thePublisher, theCopyrightYear,
            theLoanPeriod, theCallNumber, theDuration, theArtist,
            theMedium, theNumTracks));
        }

        // Precondition:  theCopyrightYear >= 0 and theLoanPeriod >= 0 and
        //                theVolume >= 0 and theNumber >= 0
        // Postcondition: A library journal has been created with the specified
        //                values for title, publisher, copyright year, loan period,
        //                call number, volume, number, discipline, and editor. The
        //                item is not checked out.
        //                The journal has been added to the Library.
        public void AddLibraryJournal(String theTitle, String thePublisher, int
theCopyrightYear,
            int theLoanPeriod, String theCallNumber, int theVolume, int theNumber,
            String theDiscipline, String theEditor)
        {
            _items.Add(new LibraryJournal(theTitle, thePublisher, theCopyrightYear,
            theLoanPeriod, theCallNumber, theVolume, theNumber,
            theDiscipline, theEditor));
        }

        // Precondition:  theCopyrightYear >= 0 and theLoanPeriod >= 0 and
        //                theVolume >= 0 and theNumber >= 0
        // Postcondition: A library magazine has been created with the specified
        //                values for title, publisher, copyright year, loan period,
        //                call number, volume, and number. The item is not checked out.
```

```
        //                The magazine has been added to the Library.
        public void AddLibraryMagazine(String theTitle, String thePublisher, int
theCopyrightYear,
            int theLoanPeriod, String theCallNumber, int theVolume, int theNumber)
        {
            _items.Add(new LibraryMagazine(theTitle, thePublisher, theCopyrightYear,
            theLoanPeriod, theCallNumber, theVolume, theNumber));
        }

        // Precondition:  None
        // Postcondition: The number of patrons in the library is returned
        public int GetPatronCount()
        {
            return _patrons.Count;
        }

        // Precondition:  None
        // Postcondition: The number of items in the library is returned
        public int GetItemCount()
        {
            return _items.Count;
        }

        // Precondition:  0 <= itemIndex < GetItemCount()
        //                0 <= patronIndex < GetPatronCount()
        // Postcondition: The specified item will be checked out by
        //                the specifed patron
        public void CheckOut(int itemIndex, int patronIndex)
        {
            if ((itemIndex >= 0) && (itemIndex < GetItemCount()))
            {
                if ((patronIndex >= 0) && (patronIndex < GetPatronCount()))
                    _items[itemIndex].CheckOut(_patrons[patronIndex]);
                else
                    throw new ArgumentOutOfRangeException($"{nameof(patronIndex)}",
patronIndex,
                        $"0 <= {nameof(patronIndex)} < GetPatronCount()");
            }
            else
                throw new ArgumentOutOfRangeException($"{nameof(itemIndex)}", itemIndex,
                    $"0 <= {nameof(itemIndex)} < GetItemCount()");
        }

        // Precondition:  0 <= bookIndex < GetItemCount()
        // Postcondition: The specified book will be returned to shelf
        public void ReturnToShelf(int itemIndex)
        {
            if ((itemIndex >= 0) && (itemIndex < GetItemCount()))
                _items[itemIndex].ReturnToShelf();
            else
                throw new ArgumentOutOfRangeException($"{nameof(itemIndex)}", itemIndex,
                    $"0 <= {nameof(itemIndex)} < GetItemCount()");
        }

        // Precondition:  None
        // Postcondition: The number of items checked out from the library is returned
        public int GetCheckedOutCount()
        {
```

```
            int checkedOutCount = 0; // Running count of checked out books

            foreach (LibraryItem item in _items)
                if (item.IsCheckedOut())
                    ++checkedOutCount;

            return checkedOutCount;
        }

        // Namespace Helper Method - Use with care
        // Precondition:  None
        // Postcondition: The list of items stored in the library is returned
        internal List<LibraryItem> GetItemsList()
        {
            return _items;
        }

        // Namespace Helper Method - Use with care
        // Precondition:  None
        // Postcondition: The list of patrons stored in the library is returned
        internal List<LibraryPatron> GetPatronsList()
        {
            return _patrons;
        }

        // Precondition:  None
        // Postcondition: A string is returned presenting the libary in a formatted
report
        public override string ToString()
        {
            // Using StringBuilder to show use of a more efficient way than String
concatenation
            StringBuilder report = new StringBuilder(); // Will hold report as being
built
            string NL = Environment.NewLine; // NewLine shortcut

            report.Append("Library Report\n");
            report.Append($"Number of items stored:      {GetItemCount(),4:d}{NL}");
            report.Append($"Number of items checked out:
{GetCheckedOutCount(),4:d}{NL}");
            report.Append($"Number of patrons stored:    {GetPatronCount(),4:d}");

            return report.ToString();
        }
    }
}
```