|  |  |  |
| --- | --- | --- |
| Name: | Blaine Simcox II | |
|  | | |
| Grade: |  | of 50 |

# Instructions

Create a new folder named “<nmcid>-final” using your own NMCID (example: *schmoej-final*). Use this folder for all exam files, including a copy of this Word document containing your name and your answers.

# (38) 3-Tier Application – Music

In this problem you will create new C# application ***Music*** to display info about orchestral instruments in database Orchestra.accdb. Please read through the entire problem description before you begin.

**Important note! Several independent C# programming concepts have been combined into this problem. If you are unable to finish one part, make sure you move ahead to implement other parts to earn their associated points.** (Examples of partial credit solutions: replace the enum with strings; pull only one field instead of two from the database; implement one of the two GetInstruments() methods.) **Make sure your completed application compiles and runs. If you can’t get a feature to work, disable it with comments and describe your problem with additional comments in your code.**

**Application configuration**

Copy database Orchestra.accdb downloaded from Moodle. Choose a relative path (best) or a fixed path (E:\Data\ or equivalent). Open the database in Access to become familiar with its schema and data. Alternative: Create and use a SQL Server database from the Access file or the XML file.

x (1 point) Create application setting orchestraConnection defining the provider and data source for the Orchestra database.

x (1 point) Create public enum Section containing four values – String, Woodwind, Brass, and Percussion. Place the definition in its own file. Use standard file naming conventions.

x (1 point) Add XML documentation for the enum and for each value within the enum – five places. Add identifying documentation at the top of the file, identifying the purpose of the enum, the programmer, and the date.

**Public individual “business” class Instrument**

Each instance of this class will correspond to a single record from the *Instruments* table.

x (1 point) Create new custom class Instrument.

x (1 point) Add string property Name to hold the name of a musical instrument (ex: “Trombone”). Add a corresponding private field. Trim leading and trailing white space.

x (1 point) Enforce Title Case.

x (1 point) Use a regular expression to ensure the value contains only letters and spaces. Throw an applicable Exception if rules fail.

xx (1 point) Add auto-generated *Section[[1]](#footnote-1)* property Category to hold the classification of the musical instrument (ex: *Section.Brass*).

x (1 point) Add a fully-specified constructor that takes two parameters: instrument’s name and category.

x (1 point) Add a default constructor. Using two constants for default values, call the fully-specified constructor from this one.

x (1 point) Override the definition of ToString().

x (1 point) Add application documentation at the top of the file, identifying the purpose of the class, the assignment, the programmer, and the date. Add several applicable comments to document your code.

x (1 point) Add XML documentation for the *Instrument* class, each property, each constructor, and the method – six total places in this file.

**Data manipulation class DataRepository**

This class encapsulates all database operations, separating the user interface from backend database-specific commands. Methods in this class return business data to the user interface class.

x (1 point) In new custom class DataRepository, add appropriate using statement(s) for working with databases (polymorphically) and for using application settings.

x (1 point) Create a DbConnection class-level variable to refer to the database. Use the connection string parameter constructor, obtaining the value from orchestraConnection.

x (1 point) Create a DbCommand class-level variable and a DbDataReader class-level variable.

x (1 point) Explain similarities and distinctions between generic ADO data provider objects such as these and their OleDb or Sql equivalents:

|  |  |
| --- | --- |
| Similarities | The Generic call is used for either SQL or OleDb the same way that you would instantiate these classes using the SQL or Access specific Class call |
| Distinctions | The SQL and OleDb data providers are file-specific whereas the Db calls are just generic. |

Add method GetInstruments(Section selectedSection) that returns a List<Instrument> collection.

x (1 point) Correctly configure the DbCommand.

x (1 point) Use a correct SQL statement to select only those records that match selectedSection, sorting results in alphabetic order by Instrument. Recommended: use an SQL parameter. Optional: use LINQ.

x(1 point) Create a new *Instrument* instance from each record returned by the DbDataReader using the fully-specified constructor. Add each to a local List<Instruments> variable.

x (1 point) Correctly cast the Category value as a member of enum Section.

x (1 point) Surround all risky database code with structured exception handling. Optional – and a good idea – also manage the iDisposable DbConnection object with a *using* statement.

Add overloaded method GetInstruments() that also returns a List<Instrument> collection. *Optional: you are welcome to implement this method by calling GetInstruments(Section selectedSection).*

x (1 point) Correctly configure the DbCommand.

x (1 point) Use SQL to select all records and to sort in ascending order by Instrument. Optional: use LINQ to sort.

x (1 point) Create a new *Instrument* instance from each record returned by the DbDataReader using the fully-specified constructor. Add each to a local List<Instruments> variable.

x (1 point) Correctly cast the Category value as a member of enum Section.

x (1 point) Surround all risky database code with structured exception handling. Optional – and a good idea – also manage the iDisposable DbConnection object with a *using* statement.

Complete the DataRepository class:

x (1 point) Add application documentation at the top of the file: identify the purpose of the class, the assignment, the programmer, and the date. Add several applicable comments to document your code. Add XML documentation for the *DataRepository* class and for each method – three places.

You are welcome to add other features if you find them helpful and appropriate. However, no other features are required in this class and none will be graded.

**User interface class**

Instantiate a *DataRepository* object.

x(2 points) Add a feature that displays the results of the *GetInstruments*() method on the form. Explain your approach:

|  |
| --- |
| BtnReset in addition to the form Initialize(), call InitializeDatabase() which in turn calls the default GetInstruments method. InitializeDatabase() is essentially an exception handling wrapper for GetInstruments() that also serves the function of resetting the GUI on each activation of the method. |

x (2 points) Add a feature that displays the results of the *GetInstruments*(*Section* *selectedSection*) method using a user-specified *Section*. Explain your approach:

|  |
| --- |
| I have a combo box on the GUI being loaded by translating the database source into the type of Enum Section via the GetType() method. The GetType() method uses the SQL statement: @"SELECT DISTINCT Type FROM Instruments" to get a list of non-repeating types to fill cboMaster on Form1.  The User’s Selection of these enums cast ToString() cause a sort using the GetInstruments(string type). |

x (2 points) Implement professional-quality style in your graphical user interface: appropriate anchoring, size management, tab order, color, font, graphics, labels, etc. List four specific and distinct design features you would like me to review for grading:

|  |  |
| --- | --- |
| The data validation enforcing title case and Regex is fully active. I purposely added lower case entries into the DB with “bass guitar” and “rhythm guitar”. They Return first letter capitalized. | Controls are all anchored and stay in place when window is resized. Image looks good for the subject matter and design is simple and user friendly. |
| RtbMaster grows to Accommodate Much larger Screen. Use of code refactoring throughout DataRepository shrinks code makes more reusable | cboType Loads from database by translating enums (see above explanation) automatically at form load. |

x (1 point) Add application documentation at the top of the file, identifying the purpose of the GUI, the assignment, the programmer, and the date. Add several applicable comments to document your code. Add XML documentation for the user interface class and for each public member.

**Wrap-up**

x (1 point) Remove unused *usings* in each file. Add region blocks around business rules, fields and class-level variables, properties, and methods as applicable. Collapse all regions. Configure an XML documentation file for your project. Rebuild your solution and resolve XML warnings, if applicable.

x (2 points) Draw a class diagram showing all classes and their relationships. Paste a graphic image here.

|  |
| --- |
|  |

**Going further with the user interface – optional**

x (1 point extra credit) Use the enum’s values (rather than design-time entered strings) to populate the choice of *selectedSection* on the GUI. For example, if you use CheckBoxes, set the Text property of the first CheckBox to be “*String*”, the second to be “*Woodwind*”, etc., by reading values from enum *Section* in the form’s constructor or in a form load event.

(1 point extra credit) Add a feature that lets the user select multiple *Sections* (such as *Brass* and *Woodwind*) at the same time. Explain your approach:

|  |
| --- |
|  |

# (12) Object-Oriented Programming Definitions

Use your own words in a concise, complete definition of each of these terms. List a situation / example for each.

(2 points) Interface

|  |  |
| --- | --- |
| Definition | A header file from C++, at least, that is all of their functionality. In C# though, an interface is a way of making objects appear to contain multiple inheritance, though the abstract class aspect of how they work really makes them nothing more than a header file like in C++. Objects that inherit interfaces can be used as both objects of the interface and as objects of whatever class inherits them. This also allows you to set constraints for each class that inherits the interface since all members of the interface must have definitions for the interface’s methods. |
| Example | Interface IDisplayable  {  String GetDiplayText(string sep);  } |
|  |  |

(2 points) Generic collection

|  |  |
| --- | --- |
| Definition | A list or another STL container that can contain Generic versions of an object. ‘T’ is the universal generic identifier for a generic list or STL object. |
| Example | Public List<T> newList = new List<T>(); |

(2 points) Namespace

|  |  |
| --- | --- |
| Definition | The container that contains an entire program, its resources, and classes etc… |
| Example | Namespace SimcoxB\_Final  {  //code goes here  } |

(2 points) Class library

|  |  |
| --- | --- |
| Definition | A project that is imported in .dll format containing functions and methods that are universally accepted among multiple applications. They are put onto a special class library file when built. Other than that, they are not much unlike static classes. |
| Example | Namespace PersonInfo  {  //class declarations go here  } |

(2 points) Nested class

|  |  |
| --- | --- |
| Definition | A class or struct inside another class. |
| Example | Class foo  {  Class IPityTheFoo  {  }  } |

(2 points) Click-Once deployment

|  |  |
| --- | --- |
| Definition | Click Once is a method that allows you to post a link to program from a website, use the uninstall features of My Computer, and check for updates each time the program is run. If you choose the publish online only option, the program will simply be run from the server where it is published. I do not have web based URL to publish to so here are the steps in the image below. |
| Example |  |

# Final Instructions

Type your name below to attest that you did not interact with other students or experts via any means during this exam, that you did not borrow solutions or hints from another student or from any prior student(s), and that you did not and will not share your solutions or any hints with other students – current or future:

|  |
| --- |
| Blaine Simcox II |

Make sure you’ve saved your completed C# application and this document in your final exam folder (the one named “<*your NMCID*>-final”). Compress the folder and submit it via Moodle.

1. Optional: if you prefer, you can substitute the nullable *Section?* instead of *Section* in this application. If you do so, you may find that you need to include a corresponding private field. [↑](#footnote-ref-1)