Case Study: Computerized Gallery Systems (CGS)

INTRODUCTION

In the projects' scenario, you are one member of a software development team responsible for the design and development of a business solution for Computerized Gallery Systems (CGS).

The CGS is a proposed computerized system for keeping track of the works of art shown and sold in a private art gallery. It includes facilities for keeping track of each work of art from the time it enters the system. This system runs on a single computer.

The current application was designed and developed using structured programming techniques. The company for which you work has decided to update existing programs using object-oriented and structured programming techniques.

YOUR ROLE

You are assigned to complete the first console prototype using an object-oriented programming language.

OBJECTIVES

The objectives of this project are:

- Write programming code based on object-oriented concepts
- Use the three structured theorem programming constructs where applicable
- Create a reusable class library to demonstrate encapsulation
- Use an abstract base to demonstrate inheritance
- Use overloading techniques to demonstrate polymorphism
- Use an event handler
- Debug and handle errors to produce an error-free application

Business Requirements Specification

General Description

The Computerized Gallery System is a proposed computerized system for keeping track of the works of art shown and stored in a local private art gallery. It includes facilities for keeping track of each work of art from the time it enters the system. It runs on a single computer.

Scope

The system deals only with pieces submitted by artists, not pieces on loan from other art institutions. Information about the piece stored is limited to the artist who created the work. Curator information is also stored.

Processing Requirements

CGS will keep track of each **piece** as it enters the gallery. This information includes the following:

- A Piece Code: each piece will be assigned a unique 5-character code.
- 40-character title of the piece.
- The ID of the artist who created the piece
- The 4-digit year in which the gallery acquired the piece.
- The estimated value
- The price the piece sold for

The system is also to keep track of the status of each piece at any time by setting a **Status** to the following values, as appropriate:

D:	On display
S:	Sold
O:	Out in storage

The following information is kept on each **artist**:

- An identifier: each artist is identified by a unique 5-character identifier.
- The artist’s name (maximum 40 characters).
- the curator ID for the artist

CGS is to store information about each **curator** who works for the gallery, as follows:

- a 5-character identifier that uniquely identifies each curator
- a 30-character name
- commission total

Each curator is to be assigned responsibility for an artist. A curator may be currently unassigned or may be assigned to one or more artists.

Functions

The following are some of the major functions identified for CGS.

Add Art Piece

The system should allow all of the information kept on a piece to be entered at this time. As well, the status of the piece should be assigned to indicate whether the piece is in storage or on display.

Sell Art Piece

The system should allow the user to update the art piece status’ to sold. It must also allow the user to find the curator associated with the artist for the piece, and award the commission to the curator.

Add Artist

The system must allow the user to add an artist and assign a curator to the artist.

Add Curator

The system must allow the user to add a curator for assignment to zero or more artists.

INITIAL PROBLEM ANALYSIS

Processes

The system should offer the user the ability to:

1. add curators
2. update information kept on each curator
3. add artists
4. receive a piece into the collection
5. sell a piece, remove it from display, and update the curator's commission
6. list artists, curators, and art pieces

ASSIGNMENT

You are one member of the team assigned to the CGS project. For this project, you must complete and supply the working prototype.

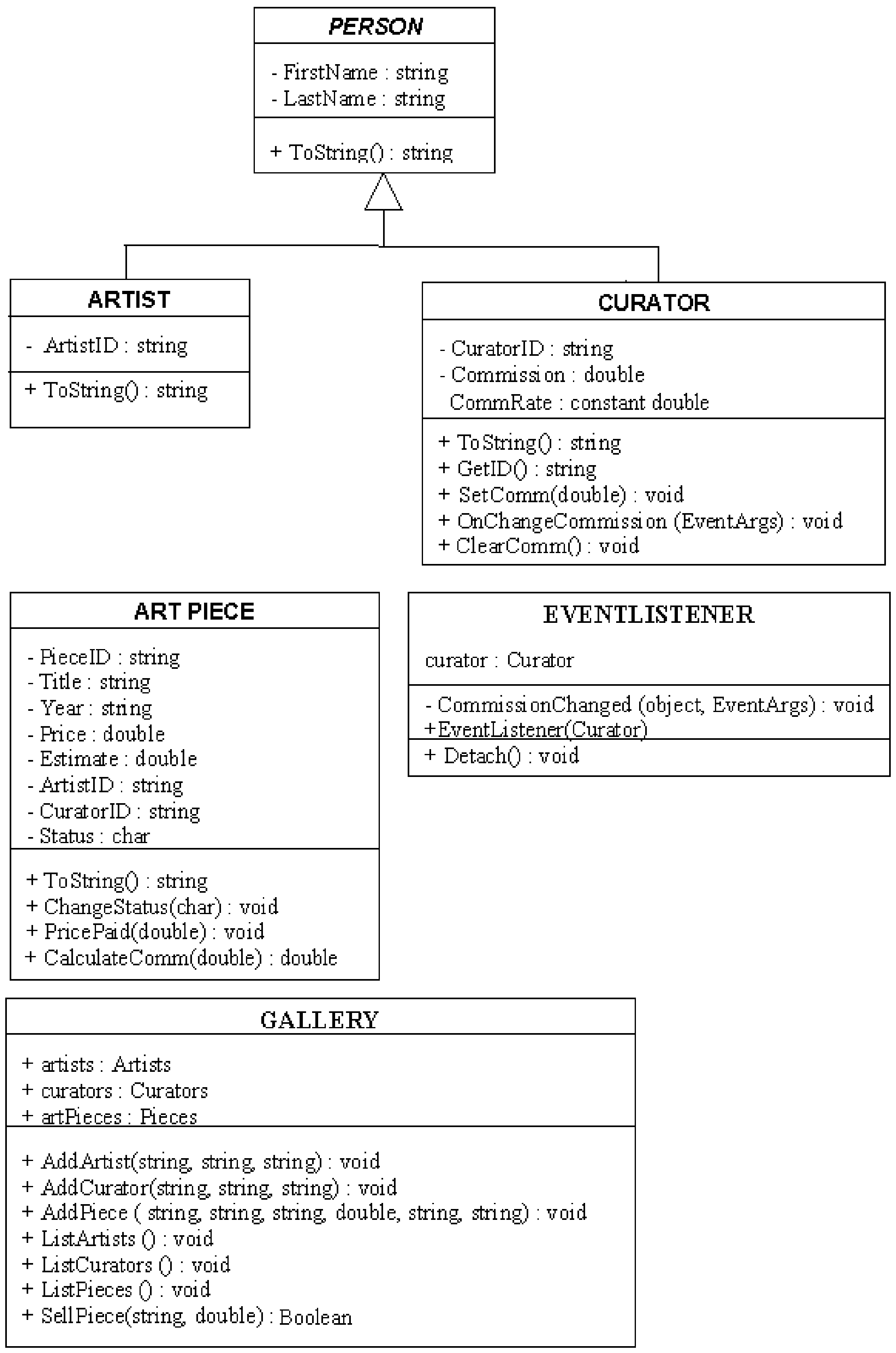
One member of the team has provided you with diagrams depicting the classes and relationships required to produce a class library for use with a client. Another member has completed the coding for classes Person, Curator, Curators, and EventListener required to create the prototype. You are responsible for classes Artist, Artists, ArtPiece, Pieces, Gallery and the ArtGallery.

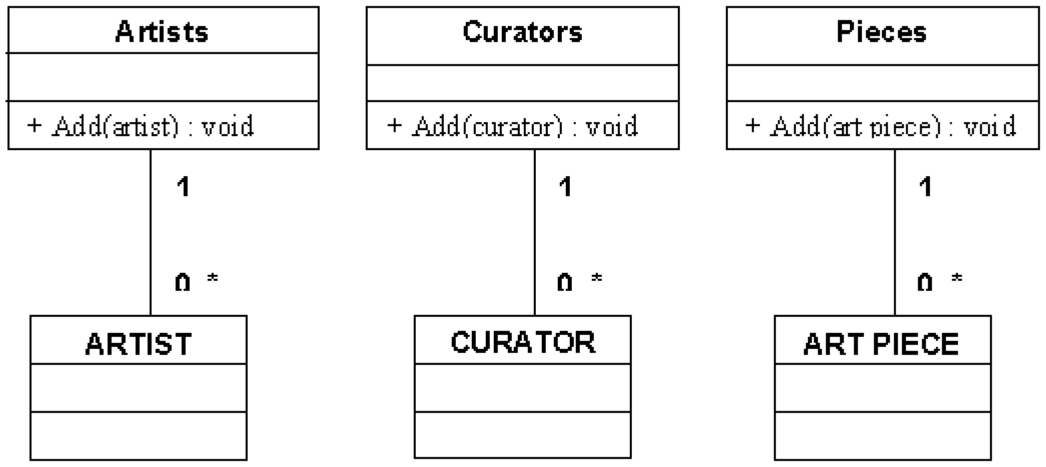
The CGS class library will encapsulate all the classes and methods so that other programmers can use it without being able to access any of the source code. It will contain the following:

- An abstract base class Person (completed)
- Two derived classes: Artist, Curator (completed)
- An ArtPiece class
- Three collection classes: Artists, Curators (completed), and Pieces
- A Gallery class through which all processing occurs
- An EventListener class for use with event handling (completed)

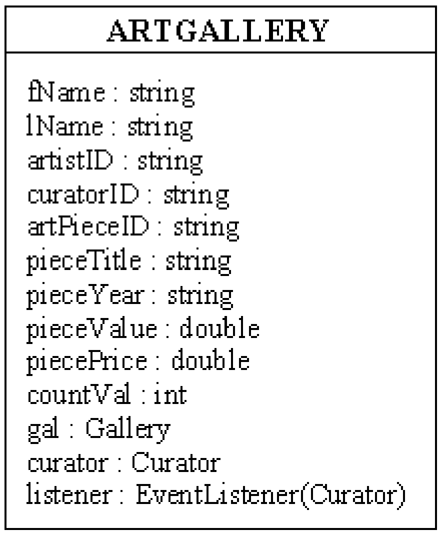
The client will contain a single class ArtGallery, which is the console application that the user would see. This client is for your test and prototype purposes. Your major responsibility is to continue to produce the class library that any programmer can use in their computerized art gallery programs.

Print out the classes already coded for you. Study the class library diagrams on the following pages and compare them to the classes already coded. Note the changes to the original specifications - Artist no longer requires the curator ID, ArtPiece requires curator ID, no 'O' status. Assume each parameter is preceded by the UML keyword 'in' (removed for clarity).


PERSON
- FirstName : string
- LastName : string
+ ToString() : string
ARTIST
- ArtistID : string
+ ToString() : string
CURATOR
- CuratorID : string
- Commission : double
CommRate : constant double
+ ToString() : string
+ GetID() : string
+ SetComm(double) : void
+ OnChangeCommission (EventArgs) : void
+ ClearComm() : void
ART PIECE
- PieceID : string
- Title : string
- Year : string
- Price : double
- Estimate : double
- ArtistID : string
- CuratorID : string
- Status : char
+ ToString() : string
+ ChangeStatus(char) : void
+ PricePaid(double) : void
+ CalculateComm(double) : double
EVENTLISTENER
curator : Curator
- CommissionChanged (object, EventArgs) : void
+EventListener(Curator)
+ Detach() : void
GALLERY
+ artists : Artists
+ curators : Curators
+ artPieces : Pieces
+ AddArtist(string, string, string) : void
+ AddCurator(string, string, string) : void
+ AddPiece (string, string, string, double, string, string) : void
+ ListArtists () : void
+ ListCurators () : void
+ ListPieces () : void
+ SellPiece(string, double) : Boolean



The following diagram illustrates the client class:



The class members are the minimum required to complete this project. You can add additional members time permitting.

You will complete the prototype in stages. First you will create the class library, and create and add classes required for the artist. Then you will then create the client application to test the artist classes. Next, you will add the classes for the curator in the class library, and test them in the client application. You will continue in this manner until the application is fully built and functional.

PROCEDURE

Sketch a flowchart or write pseudocode for all structured logic. You need to hand in your sketches. Make sure they are legible.

STEP 1:

1. Create a class library named CGS. Copy the classes supplied to you into the folder automatically created for CGS on your hard drive.
2. Add the existing class Person to the CGS class library.
    a. To add an existing file to a project in Visual Studio .NET, select and right-click the project name in the **Solution Explorer**. From the popup menu choose **Add**, **Add Existing Item** and browse to the file, in this case the Person class in the CGS folder. Click **OK**.

    b. The ToString method returns a single string containing the first and last names.
3. Add the new class Artist to the CGS class library.
    a. Add the class as a separate file. Make sure the namespace is CGS. All library class files must be in the CGS namespace.

    b. Make sure Artist inherits from Person.

    c. You can leave in the default constructor if you wish.

    d. Add accessors for the private property members.

    e. The ToString method overrides the base class method and should return a single string containing the artist's full name, and ID.

    f. Build the CGS library.

    Depending upon the machine, building may take some time. Typically, you proof read your work and correct any errors before building the library to avoid unnecessary delay.

    g. Correct any errors, test and save when CGS compiles error-free.
4. Add the new Artists class to the CGS class library.
    a. The Add method should call the List.Add() method in order to add an artist to the Artists collection.

    b. Use an indexer implementation (List collection).

    c. Build the CGS library. Correct any errors.
5. Before continuing to add more classes to the library, it is a good idea to check if the classes you have already added will run. To do this, you need to start work on the Gallery class.

    Add the new class Gallery to the CGS library.

    a. Add a new Artists instance to the Gallery class.

   b. Add the method AddArtist. This method will receive the first name, last name and ID from the client and pass them along to the Artist constructor. AddArtist uses the Artists Add method to create a new Artist instance and add it to the Artists collections.
   c. Add the method ListArtists. This method iterates through the Artists list, if any exist, appending each artist's information to one local string, which is returned to the client.
   d. Build the CGS library. Correct any errors.
6. The next step is to create the console client so you can run the application to check the work you have completed so far.
   a. Save and close CGS class library.

STEP 2:

1. Create a new console application named ArtGallery. This application is the interface that the user would see.
   a. Add the project reference CGS.dll to the solution. The dll file should be in CGS\bin\debug.
   b. Add the first three variables illustrated in the client class diagram above to Main.
   c. Create the new Gallery instance named gal.
   d. Prompt the user to enter the artist's first name. Save the user's response to the local fName variable. Do the same for the artist's last name and ID.
   e. Call the Gallery AddArtist method and pass the three values obtained from the user.
   f. Call the Gallery ListArtist method and write the results to the console.
   g. Build the ArtGallery console client. Correct any errors.
   h. Save your work.
2. Run ArtGallery without debugging to check that the CGS library created thus far does run.
   a. If an error that originates in the CGS class library occurs, you cannot correct it while in the ArtGallery client console application.
      i. You must close the ArtGallery, and then open the CGS library to correct the error.
      ii. Once you have made the correction in the CGS library, build the library again, save and close it.
      iii. Open the client console ArtGallery and build it. Run the application to see if the library error is corrected. If it is not,

you must close ArtGallery and re-open the library. (Sometimes it is difficult to tell which application you are in. Always refer to the title at the top of the window and to the name of the solution in the Solutions window.)

3. Once you are satisfied that the code you have written so far works, close ArtGallery, and open CGS library to continue adding classes.

STEP 3:

1. Add the existing class Curator to the CGS library.
    a. Check that the namespace is CGS.
    b. Curator inherits from Person.
    c. The CommRate is 0.10.
    d. In the constructor, the default value for commission is 0. Only the name and id values are passed into the constructor.
    e. The method ToString overrides the base class method and will return the curator's first and last names, the id and the total commissions earned.
    f. For the GetID method, the ID is returned to the SellPiece method in Gallery so that it can identify the ArtPiece curator for whom commission should be calculated.
    g. The SetComm method receives the amount eligible for commission for an art piece (this amount is determined by the CalculateComm method in ArtPiece, described in the next step). SetComm uses CommRate to calculate the 10% commission due and assigns it to the curator identified by the ArtPiece.
    h. Ignore the remaining methods for now. Do not uncomment the code. You will return to Curator later to add an event.
    i. Build the library, and correct any errors.
2. Add the existing Curators class. Build and correct any errors.
3. Edit the Gallery class:
    a. Include the new Curators instance.
    b. Create the AddCurator method to pass the first and last names, and the curator id along to the Curator constructor.
    c. Create the ListCurators method.
4. Build the library, and correct any errors. Save and close CGS library.
5. Open ArtGallery.

    a. Add the curator ID variable. You do not need to add variables for the curator’s first and last names. You can use the name variables that already exist.
    b. Add code to prompt the user for the curator information. Pass the information in Gallery AddCurator method.
    c. Add code to call Gallery’s ListCurators method and output the results to the screen.
    d. Build, and run the application. If any class library errors occur, correct them as instructed previously in STEP 2, #2.
    e. Close ArtGallery.

STEP 4:

1. Open CGS library.
2. Add the new ArtPiece class to the library.
    a. The constructor does not receive the price or the status. Set the default for price to 0, and the Status to ‘D’.
    b. The ToString method should return in a single string all information about the art piece. (HINT: Make sure all values are string type.)
    c. The ChangeStatus method changes the status to ‘S’ when an art piece is sold.
    d. PricePaid receives and assigns the price paid when an art piece is sold.
    e. The CalculateComm method receives the price paid. It returns 25% of the difference between the original value and the price paid. This value is sent to the SetComm method in the curator instance.
    f. Build, and correct any errors.
3. Add the new Pieces collection class. Add the Add and indexer methods. Build and correct any errors.
4. Edit the Gallery class:
    a. Add a new Pieces instance.
    b. Add methods AddPieces and ListPieces.
    c. Add the method SellPiece, which receives the ID of the art piece being sold and the price paid. This method iterates through the Pieces to find the piece with the ID matching that was passed in. Once found, the method checks the current status, if it is already sold, the method returns false. If the status is on display, it can be sold and the method extracts the Curator ID assigned to the art piece using the Curator method GetID, and calls the ArtPiece’s ChangeStatus, PricePaid and CalculateComm methods.

    The method uses the Curator ID returned by GetID to find the curator by iterating through the curators. If found, the price is passed to the Curator SetComm method.

    d. Build, correct, save and close the CGS library.

5. Open the ArtGallery.
    a. Add the code to prompt the user to enter information for the art piece. The ArtPiece constructor assigns default values to status and price therefore you do not need to prompt for them.
    b. Pass the information to create an art piece via Gallery's AddPieces method.
    c. Call the ListPieces method and output the results to the screen.
    d. Prompt the user for the ID and the price paid for the piece being sold.
    e. Pass the information via SellPiece (Price must be a double data type). SellPiece will return a Boolean value for you to test. If it is false, the sale was not completed.
    f. Call the ListPieces and ListCurators methods to verify the changes have been made.
6. Build, correct and run the application. Correct any library errors in the library class.
7. Close ArtGallery.

STEP 5:

The library is fine as it is but it would be nice to add a feature that verifies the curator is paid commission. You will add the event that raises the message 'The curator was paid commission.' in the client.

1. Open CGS library, and add the existing class EventListener.
    a. The EventListener includes a new Curator instance. The constructor receives a Curator instance. It assigns the received instance to the local Curator instance and attaches the curator's Changed event in the constructor.
    b. The CommissionChanged event has object and EventArgs type parameters. When called, this method outputs to the console screen a message informing the user that the curator was paid commission.
    c. The method Detach sets the curator to null.
    d. Build, and correct the library.
2. Edit Curator:

    a. In the CGS namespace, uncomment the code that declares a delegate event handler CommissionPaidHandler with object and EventArgs type parameters.
    b. In the Curator class:
        1. Uncomment the code that declares the event Changed of the delegate type.
        2. EventListener creates a curator instance. The overloaded constructor receives the first and last names, the curator ID and the commission.
        3. Uncomment the virtual method OnChangeCommission with an EventArgs type parameter. This method checks if Changed is null. If Changed is not null, it passes the current curator instance, and an event argument.
        4. Uncomment the method ClearComm to set commission to 0. It calls OnChangeCommission and passes empty EventArgs parameter.
        5. You also need to edit the SetComm method. Uncomment the code that calls OnChangeCommission and passes empty EventArgs parameter immediately following the final line of code which updates the curator's commission.
3. Close the CGS library and open the ArtGallery client.
    a. Add a new curator instance and a new listener instance.
    b. After the call to SellPiece, add a call to curator's ClearComm method. At the end of the code, add a call to listener's Detach method.
4. Build, correct and run the application. Debug until the application runs as it should. Save.

STEP 6:

1. The ArtGallery interface is not very useful or attractive. You will add a menu for users. Sketch a quick algorithm to help guide you in coding.
2. Add a menu, prompt for choice and switch statement within a loop. Be sure to add a Quit choice. Use the return statement to exit the program in response to the Quit selection. Edit Main to return an integer value.
3. Create methods for the switch statement to call:
    a. Create a NewArtist method by enclosing the code prompting the user for artist information and the call to the AddArtist.
    b. Create a NewCurator method by enclosing the code prompting the user for curator information and the call to the AddCurator.

   c. Create a NewArtPiece method by enclosing the code prompting the user for art piece information and the call to the AddPiece.
   d. Create a SellArtPiece method by enclosing the code prompting the user for the sale information and the call to the SellPiece.
   e. You can create methods for each of the list calls, or instead call them directly from inside the switch statement.
4. Build, correct and run the application. Debug until the application runs as it should. Save.

STEP 7:

1. Add appropriate error-handling in both the CGS library and ArtGallery.
2. Save your work for submission.
3. Keep a copy of the CGS library for use in the project in which you will create a Windows application.

If you have time

You can add to your application's functionality. Make sure you keep a copy of your completed thus far before you attempt any additional work.

1. In ArtGallery, add the logic to validate user input to ensure numeric values when required.
2. Add one or more of the following methods to the library:

   FindCurator(string) : string

   This method belongs to the Gallery class. It receives the curator's ID, and iterates through the curators comparing the IDs to the value. If the ID is found, it calls the ToString method in Curator otherwise it displays a message stating the curator does not exist.

   Add a prompt in ArtGallery for an ID to pass into FindCurator.

   FindArtist(string) : string

   This method belongs to the Gallery class. It receives the artist's ID, and iterates through the artists comparing the IDs to the value. If the ID is found, it calls the ToString method in Artist otherwise it displays a message stating the artist does not exist.

   Add a prompt in ArtGallery for an ID to pass into FindArtist.

   DeleteCurator(string) : string

   This method belongs to the Gallery class. It receives the curator's ID, and iterates through the curators comparing the IDs to the value passed in. If the ID is found, it calls the Remove method in Curators otherwise it displays a message stating the curator does not exist.

   Add a prompt in ArtGallery for an ID to pass into DeleteCurator.

   You need to add the Remove method to the Curators collection class.

   DeleteArtist(string) : string

   This method belongs to the Gallery class. It receives the artist's ID, and iterates through the artists comparing the IDs to the value. If the ID is found, it calls the Remove method in Artist otherwise it displays a message stating the artist does not exist.

   Add a prompt in ArtGallery for an ID to pass into DeleteArtist.

   You need to add the Remove method to the Artists collection class.