Course Title	Programming Techniques
Project Code	PTCE v1-0 Project 1 Part 1
Project Title	Computerized Gallery System (CGS)
Pages	15 pages (plus cover sheet)
Released	
Revised	25-04-2016

Programming Techniques

Project 1 Part 1: Computerized Gallery Systems (CGS)

INTRODUCTION

This project is the first part of the last in a series of four, which you will complete in the Phase 1 courses. 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

TIME REQUIRED

You will require 15 hours to complete this project.

MATERIALS REQUIRED

To complete this project, you require:

Hardware

- One PC per student with an Internet connection and access to a printer
- Processor 600 MHz (1G MHz recommended)
- Windows 2000 (with SP 4), XP Professional (with SP 2) or Windows Server 2003 (with SP 1) operating system

- 192 MB RAM (256 MB RAM recommended)
- 2G hard disk space (minimum) for the operating system and applications
- DVD-ROM
- 800 x 600 (1024 x 768 recommended) SVGA monitor
- 1 blank diskette

Software

- Microsoft Office Professional Edition
- Microsoft Visual Studio 2010 Professional
- AVG Anti-virus (or similar antivirus software)
- Microsoft Internet Explorer version 9 or later

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.

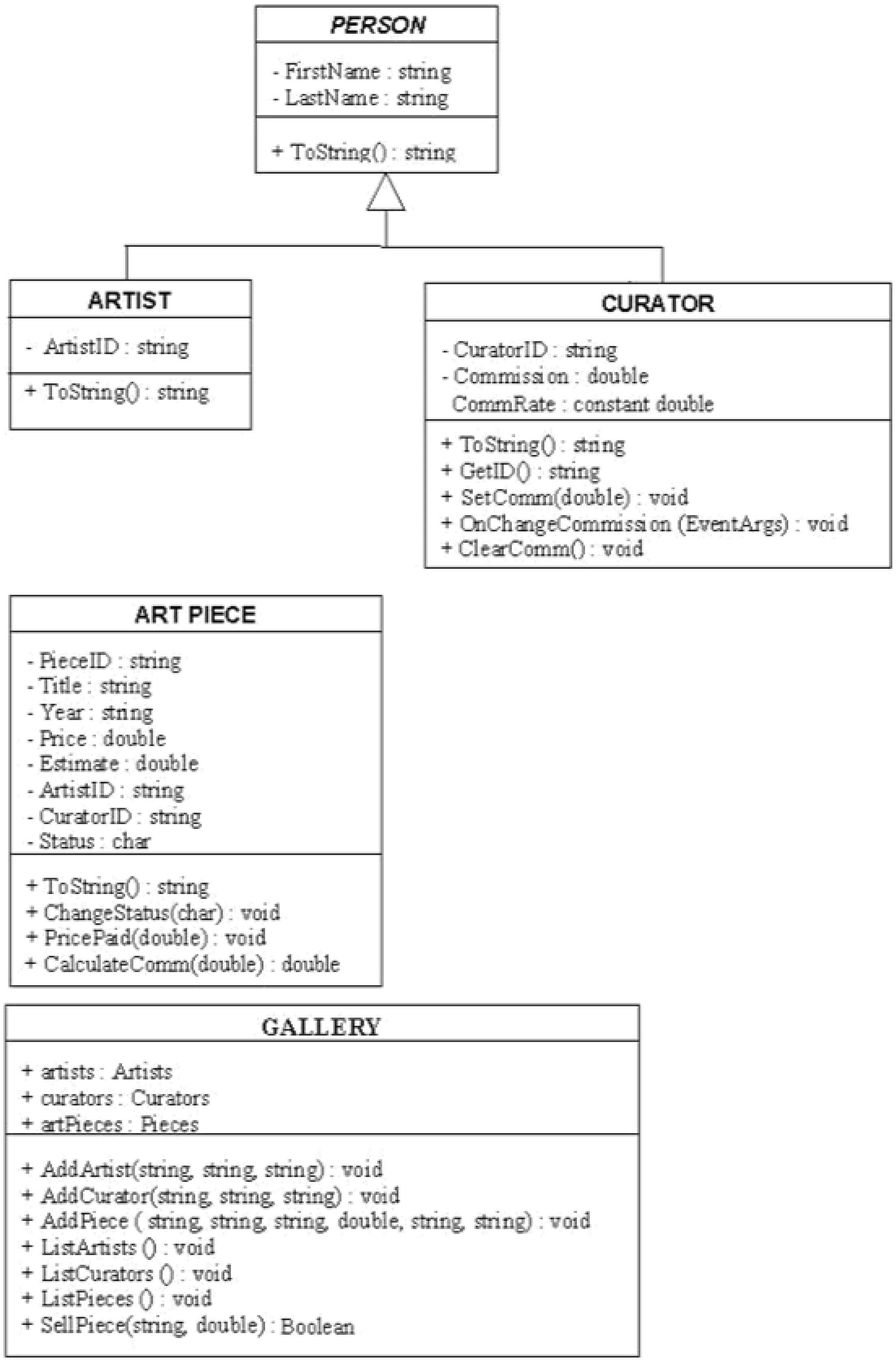
One member of the team has provided you with diagrams depicting the classes and relationships required to produce a client application. Another member has completed the coding for classes Person, Curator required to create the prototype. You are responsible for classes Artist, ArtPiece, Gallery and the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

Your application will contain the following:

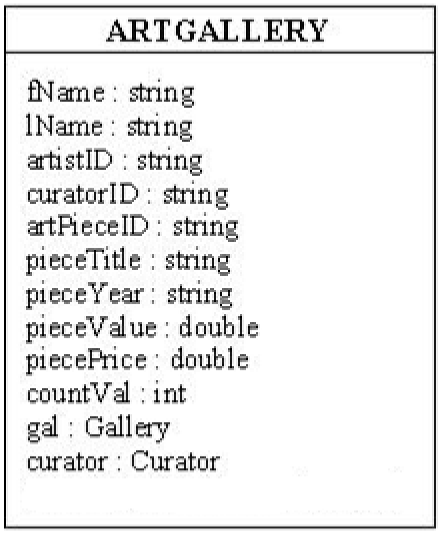
- An abstract base class Person (completed)
- Two derived classes: Artist, Curator (completed)
- An ArtPiece class
- A Gallery class through which all processing occurs

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 different classes.

Print out the classes already coded for you. Study the class 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
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:



ARTGALLERY

fName : string
lName : string
artistID : string
curatorID : string
artPieceID : string
pieceTitle : string
pieceYear : string
pieceValue : double
piecePrice : double
countVal : int
gal : Gallery
curator : Curator

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. Add the existing class Person to your project.
    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.
2. Add the new class Artist.
    a. Make sure Artist inherits from Person.
    b. You can leave in the default constructor if you wish.
    c. Add accessors for the private property members.
    d. The ToString method overrides the base class method and should return a single string containing the artist's full name, and ID.
    e. Build your application.
    f. Correct any errors, test and save when CGS compiles error-free.
3. Add the Artists collection to the Gallery class : (Use ArrayList or List< >)
    a. 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 Add method of **List** to create a new Artist instance and add it to the Artists collections.
    b. 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.
    c. Build the Project. Correct any errors.
4. The next step is to create the console client so you can run the application to check the work you have completed so far.

STEP 2:

1. Create a new console application named ArtGallery. This application is the interface that the user would see.
    a. Add the first three variables illustrated in the client class diagram above to Main.
    b. Create the new Gallery instance named gal.
    c. 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.
    d. Call the Gallery AddArtist method and pass the three values obtained from the user.
    e. Call the Gallery ListArtist method and write the results to the console.
    f. Build the ArtGallery console client. Correct any errors.
    g. Save your work.

STEP 3:

1. Add the existing class Curator to the Project.
    a. Curator inherits from Person.
    b. In the constructor, the default value for commission is 0. Only the name and id values are passed into the constructor.
    c. 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.
    d. 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.
    e. 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.
    f. Build the project, and correct any errors.

2. 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.
    d. 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.
    e. Add code to prompt the user for the curator information. Pass the information in Gallery AddCurator method.
    f. Add code to call Gallery’s ListCurators method and output the results to the screen.

STEP 4:

1. Add the new ArtPiece class to the project.
    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.
2. Edit the Gallery class:
    a. Add a new Pieces instance.
    b. Add the new Pieces collection : (Use ArrayList or List< >)
    c. Add methods AddPieces and ListPieces.
    d. 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.

3. Edit the Programm class:
    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.

4. Build, correct and run the application. Correct any library errors in the library class.

STEP 5:

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 6:

1. Add appropriate error-handling in the ArtGallery.
2. Save your work for submission.

STEP 7:

1. In your Conclusion, write a short paragraph explaining how OO concepts of encapsulation, inheritance and polymorphism are accomplished in the project. Describe the difference between a structured program and an object-oriented program.

Congratulations!

You have created your first object-oriented application complete with a reusable class library.

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.

Marking Scheme

You are graded on the following components:

Project component	Points
Construction of flowchart and/or pseudocode	**5**
Use of encapsulation, inheritance and polymorphism	**10**
Use of collection classes	**5**
Use of private and public data member functions, accessor functions	**10**
Use of constructors and overloaded constructors	**10**
Use of Event Handler and event	**10**
Proper code, format, and appropriate comments within the program	**5**
The program functions successfully with the desired output	**20**
Client (console) interface functions as required for successful output	**8**
Debugging and error-handling to produce a virtually error-free application	**7**
Documentation and presentation including the conclusion	**10**
Total Number of Points Possible:	**100**

What to Submit

Your project must contain:

- Title Page
- Project Description
- Print-out of program written in C# interpreting the algorithms you developed and the classes' design
- Diskette containing the ArtGallery client prototype written in C#
- A list detailing clearly what user input is validated (if any), and what input is not. This helps your instructor determine what type of input to enter to run and evaluate your project properly.
- Sketches of flowchart algorithms and/or pseudocode
- Conclusion

Penalties

- For each day that a project is late, 5% will be deducted.
- Projects that are more than three days late will earn a maximum score of 60%.
- Projects that contain a virus must be resubmitted and will earn a maximum of 60%.