Case Study: Electronic Telethon Systems (ETS)

INTRODUCTION

This project is part one in a series of three, which you will complete in this semester. In the projects' scenario, you are one member of a software development team responsible for the design and development of a business solution for Electronic Telethon Systems (ETS).

The ETS tracks donations called in by telephone during a telethon. The system keeps track of each donation, each donor, and the allocation of prizes. Every donation is paid in full by credit card at the time it is taken. 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 Electronic Telethon System (ETS) tracks donations called in by telephone during a telethon. The system keeps track of each donation, each donor, and the allocation of prizes. Every donation is paid in full by credit card at the time it is taken. It runs on a single computer.

Processing Requirements

The ETS will keep track of each **donation**. The information kept about each donation includes the following items:

- Donation Identifier —- Each donation will be assigned a unique 4-character code.
- Donation Date
- Donor Identifier — Each donor will be kept track of in a master file, keyed on a unique donor identifier. The donation will refer to this donor ID.
- Donation Amount (smallest value: $5.00; largest value: $999,999.99)
- Prize ID

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

- 4-character Donor ID
- 15-character Donor's Last Name
- 15-character Donor's First Name
- Address of 40 characters
- Phone Number (999 999-9999)
- Credit Card Type (1-character code)
- 16-character Credit Card Number
- Credit Card Expiry Date (mm/yyyy)

One donor may make several separate donations.

The following information is kept on each **prize** (**Note:** A prize refers to a *type* of item that may be won by donors. The system only has to keep summary information on types of prizes, not on each physical item):

- Prize ID — Each type of prize is identified by a unique 4-character prize identifier. For example, telethon mugs might be represented by Prize ID: M132.
- 15-character Prize Description
- Prize Value — The value of 1 prize item. Maximum value: $299.99.
- Donation Limit – The minimum donation required to award the prize
- Quantity (# Items) Still Available — Maximum value 999
- Quantity (# Items) Originally Available
- Sponsor ID

The **quantity originally available** is set when the prizes arrive at the telethon offices from the sponsor who sent them. (More on sponsors later.) Each prize type may be awarded to a number of donations. For example, coffee mugs could be awarded to two separate donations as follows: one donation receives an award of five coffee mugs and another donation receives an award of two coffee mugs; however, a single donation may receive none or one prize types. When a prize is awarded, the **quantity still available** for that prize should be reduced by the appropriate amount.

As well, the system must keep track of **sponsors**. Each sponsor is identified by a 4-character identifier. The system must know the name of each sponsor (30 characters) and the total value of the prizes the sponsor has provided, and it should be able to track which prizes have been provided by which sponsors. A **prize**, such as coffee mugs, is supplied by exactly one **sponsor**. One sponsor, however, may supply a number of different types of prizes.

Initial Problem Analysis

The system should offer the user the ability to:

1. add sponsors and the prizes they provide
2. add donors and the donations they make
3. award a prize for a donation, and update the number of prizes still available

ASSIGNMENT

You are one member of the team assigned to the ETS 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, Sponsor, Sponsors, Prize, Prizes and EventListener required to create the prototype.

The ETS 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 should contain the following:

- An abstract base class Person
- Two derived classes: Sponsor, Donor
- A Donation class
- A Prize class
- Four collection classes: Sponsors (completed), Donors, Donations and Prizes
- An ETSManager class through which all processing occurs
- An EventListener class for use with event handling

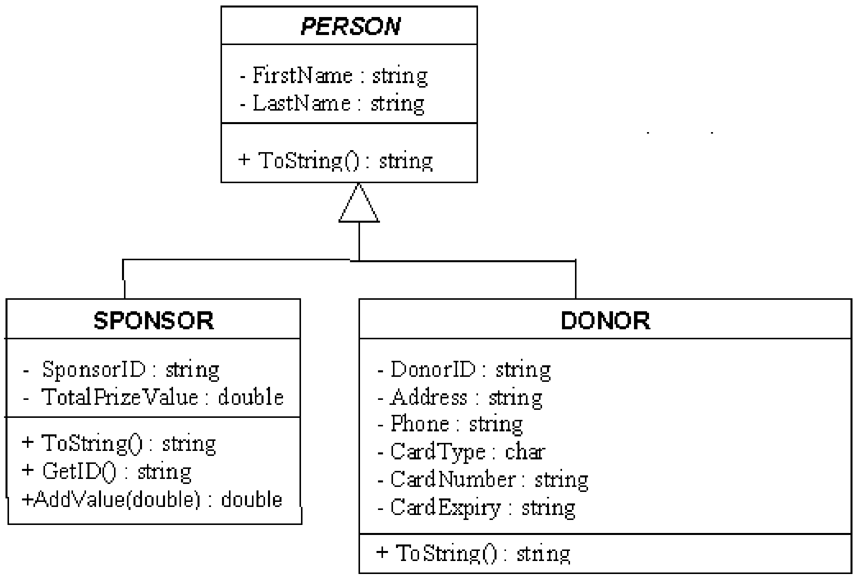
The client will contain a single class TelethonSystem, which is the console application that the user would see. This client is for your test and prototype purposes. Your major responsibility is to continue to produce the class library that any programmer can use in their electronic telethon system programs.

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:

> This prototype requires you to add and list sponsors, add and list prizes, add and list donors, add and list donations, and award a prize.

Assume each parameter is preceded by the UML keyword ‘in’ (removed for clarity).

You are responsible for creating classes Donor, Donors, Donation, Donations, and ETSManager and the client TelethonSystem.


PERSON
- FirstName : string
- LastName : string
+ ToString() : string
SPONSOR
- SponsorID : string
- TotalPrizeValue : double
+ ToString() : string
+ GetID() : string
+AddValue(double) : double
DONOR
- DonorID : string
- Address : string
- Phone : string
- CardType : char
- CardNumber : string
- CardExpiry : string
+ ToString() : string

PRIZE

- PrizeID : string
- Description : string
- Value : double
- DonationLimit : double
- OriginalAvailable : int
- CurrentAvailable : int
- SponsorID : string

+ ToString() : string
+ GetPrizeID() : string
+ Decrease(integer) : void
+ OnChangePrize(EventArgs) : void
+ Clear Prize () : void

DONATION

- DonationID : string
- DonationDate : string
- DonorID : string
- DonationAmount : double
- PrizeID : string

+ ToString() : string

EVENTLISTENER

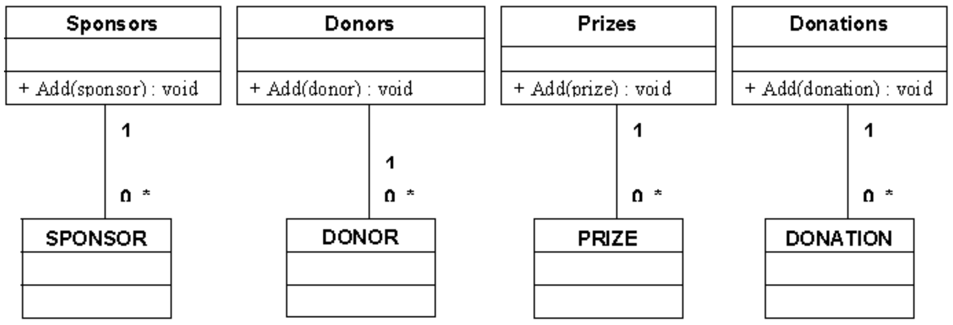
prize : Prize

- PrizeAwarded (object, EventArgs) : void
+EventListener(Prize)
+ Detach() : void

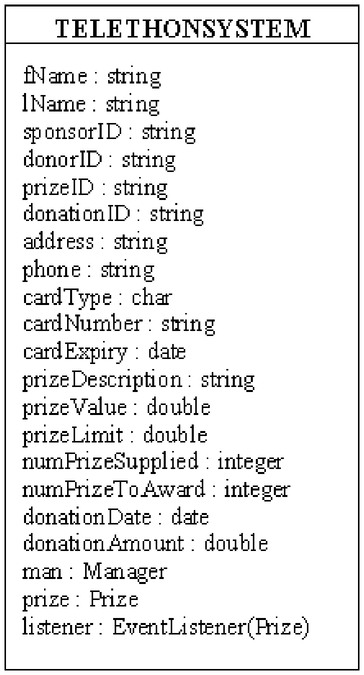
ETSMANAGER

+ donors : Donors
+ sponsors : Sponsors
+ donations : Donations
+ prizes : Prizes

+ AddDonor(string, string, string, string, string, char, string, date) : void
+ AddSponsor(string, string, string, double) : void
+ AddPrize (string, string, double, double, double, string) : void
+ AddDonation(string, date, string, double, string) : void
+ ListDonors () : string
+ ListSponsors () : string
+ ListPrizes ( ) : string
+ ListDonations () : string
+ ListQualifiedPrizes(double) : string
+ RecordDonation(string, integer, string, string, string) : Boolean



The following diagram illustrates the client class:



```
TELETHONSYSTEM

fName : string
lName : string
sponsorID : string
donorID : string
prizeID : string
donationID : string
address : string
phone : string
cardType : char
cardNumber : string
cardExpiry : date
prizeDescription : string
prizeValue : double
prizeLimit : double
numPrizeSupplied : integer
numPrizeToAward : integer
donationDate : date
donationAmount : double
man : Manager
prize : Prize
listener : EventListener(Prize)
```

Note: In the above class, cardExpiry and donationDate should be string data types.

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 add the person, sponsor and prize classes to it. Then you will create the client application to test the library containing the sponsor and prize classes. When the prototype completed thus far works, you will create the donor, donors, donation and donations classes 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

STEP 1:

1. Create a class library named ETS. Copy the classes into the folder automatically created for ETS on your hard drive.
2. Add the existing Person class to the ETS 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 ETS folder. Click **OK**.
    b. The ToString method returns a single string containing the first and last names.
3. Add the existing Sponsor class to the ETS class library.
    a. The namespace is ETS. All library class files must be in the ETS namespace.
    b. Sponsor inherits from Person.
    c. The ToString method overrides the base class method and returns a single string containing the sponsor's full name, and ID.
4. Add the existing Sponsors class to the ETS class library.
    a. The Add method calls the List.Add() method to add an sponsor to the Sponsors collection.
    b. Sponsors class uses an indexer implementation (List collection).
5. Add the existing Prize class to the ETS class library.
    a. Ignore all code to manage the event handler for now. Do not uncomment the code.
    b. The ToString method overrides the base class method and returns a single string containing the prize ID, description, donation limit and number of prizes currently in stock.
    c. When a new prize is supplied, the quantity supplied is also assigned to the current quantity available.
    d. The Decrease method subtracts the number of prizes awarded from the quantity available to arrive at the new number of prizes available.
6. Add the existing Prizes class to the ETS class library.
    a. The Add method calls the List.Add() method to add a prize to the Prizes collection.
    b. Prizes class uses an indexer implementation (List collection).

7. Build the ETS 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.
8. Before continuing to add more classes to the library, it is a good idea to check the classes you have already added work. To do this, you need to start work on the ETSManager class.

   Add the new class ETSManager to the ETS library.
9. Add a new Prizes instance to the ETSManager class.
    a. Add the method AddPrize. This method will receive the prize ID, description, value for a single prize, the donation limit, the number of prizes supplied and the sponsor ID to pass them along to the Prize constructor. AddPrize uses the Prizes Add method to create a new Prize instance and add it to the Prizes collections.
    b. Add the method ListPrizes. This method returns the string(s) that contains prize information from Prizes. It iterates through the Prizes list, if any exist, appending the information to one local string, which is returned to the client.
    c. Build the ETS library. Correct any errors.
10. Add a new Sponsors instance to the ETSManager class.
    a. Add the method AddSponsor. This method will receive the first name, last name, ID and total value of prizes supplied from the client (multiply the number of prizes input by the value of each prize) to pass them along to the Sponsor constructor. AddSponsor uses the Sponsors Add method to create a new Sponsor instance and add it to the Sponsors collections.
    b. Add the method ListSponsors. This method returns the string(s) that contains sponsor’s name and ID from Sponsors. It iterates through the Sponsors list, if any exist, appending the information to one local string, which is returned to the client.
    c. Build the ETS library. Correct any errors.
11. The next step is to create the console client so that you can run the application to test the work you have completed so far.
    a. Save and close ETS class library.

STEP 2:

1. Create a new console application named TelethonSystem. This application is the interface that the user will see.
    a. Add the project reference ETS.dll to the solution. The dll file should be in ETS\bin\debug.
    b. Declare the sponsor and prize variables listed in the client class diagram in Main.

   c. Create the new ETSManager instance named **man**.
   d. Prompt the user to enter the sponsor’s first name. Save the user’s response to the local fName variable. Do the same for the sponsor’s last name and ID.
   e. Prompt the user to enter the prize information: ID, description, value, number of prizes supplied, and the donation limit.
   f. Call the ETSManager’ AddSponsor method and pass the four values obtained from the user (don’t forget to calculate the total value for all prizes supplied).
   g. Call the ETSManager’ AddPrize and pass the six values obtained from the user (this includes the sponsor ID).
   h. Call the ETSManager’ ListSponsors method and write the results to the console.
   i. Call the ETSManager’ ListPrizes method and write the results to the console.
   j. Build the TelethonSystem console client. Correct any errors.
   k. Save your work.

2. Run TelethonSystem without debugging to check that the ETS library created thus far does work.
   a. If an error that originates in the ETS class library occurs, you cannot correct it while in the TelethonSystem client console application.
      i. You should close the TelethonSystem, and then open the ETS library to correct the error.
      ii. Once you have made the correction in the ETS library, build the library again, save and close it.
      iii. Open the client console TelethonSystem and build it. Run the application to see if the library error is corrected. If it is not, you should close TelethonSystem 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 TelethonSystem, and open ETS library to continue adding classes.

STEP 3:

~~A~~dd a new class to the ETS library. Name the class Donor. Make sure the namespace is ETS.
   a. After you open the Donor class file, check that the namespace is ETS. If it is not, you must edit it to ETS.

    b. Make sure Donor inherits from Person.
    c. Add the class constructor, private members and accessors.
    d. Add the method ToString. It overrides the base class method and will return the donor’s first and last names, the id, address, phone number and the credit card information.
    e. Build the library, and correct any errors.

2. Create the Donors class. Add the Add and indexer methods. Build and correct any errors.
3. Create the Donation class in the ETS library.
    a. Add the class private members and accessors and ToString method.
    b. The constructor should assign today’s date for the donation date.
    c. Build the library, and correct any errors.
4. Create the Donations class. Add the Add and indexer methods. Build and correct any errors.
5. Edit the ETSManager class:
    a. Include the new Donors instance.
        1. Create the AddDonor method to pass the first and last names, id, address, phone, cardtype, cardnumber and card expiry along to the Donor constructor.
        2. Create the ListDonors method.
        3. Build the library, and correct any errors.
    b. Include the new Donations instance.
        1. Create the AddDonation method to pass along the donation ID, donorID, prizeID, and donation amount to the Donation constructor.
        2. Create the ListDonations method.
        3. Build the library, and correct any errors.
    c. Add the method ListQualifiedPrizes to ETSManager. This method lists only those prizes, for which the donation limit is less than or equal to the donation made by the donor.
    d. Add the method RecordDonation to ETSManager. This receives the prize id selected by the user, number of prizes awarded by the user, the donor’s id, and the donation amount. It ensures a prize can be awarded for a donation. If the prize cannot be awarded, the donation is not made.
        1. The method iterates through the prizes to find the matching prize id. If found, and if the current number of prizes available

is greater than or equal to the number awarded by the user, the number of prizes currently available is reduced by the number of prizes awarded (use the prize Decrease method).

2. If the prize ID was found and there were enough prizes to award, the prize ID is stored so that the method can call AddDonation.
3. The method returns true if the donation was added, and false if it was not.

6. Build the library, and correct any errors. Save and close ETS library.
7. Open TelethonSystem.
    a. Add local variables for the prize and donor.
    b. Add code to prompt the user for the donor information.
    c. Add code to prompt the user for the donation information.
    d. Call the ListQualifiedPrizes method in ETSManager, and prompt the user for the prize ID and number of prizes to award for the donation.
    e. Call the RecordDonation method. If it returns true, the donation was made and the prize was awarded (HINT: you will need to use a flag variable to hold the true or false value).
    f. If the method returned false, the system was unable to award the prize and the donation was not made. Inform the user through the Console.
    g. To allow the user another chance to select a prize, enclose the code starting with the instruction to write ListQualifiedPrizes to the console to the instruction to write the unable to add donation message to the console inclusive. The loop exits if the donation was added successfully, or if the user does not wish to continue. Add code to prompt the user if they wish to continue looking if they cannot find a suitable prize.
    h. If RecordDonation returned true, call the AddDonor method. The donor is not added before the call to RecordDonation because a prize may not be available.
    i. Add code to call ETSManager's ListPrizes method and output the results to the screen.
    j. Build, and run the application. If any class library errors occur, correct them as instructed previously in STEP 2, #2.

STEP 4:

The library is fine as it is but it would be nice to add a feature that verifies the prize is awarded to the donor. You will add the event to raise the message 'The prize has been awarded to the donor' in the client.

1. Add the existing class EventListener to the ETS library.

a. The EventListener includes a new Prize instance. The constructor receives a Prize instance. It assigns the received instance to the local Prize instance, and attaches the prize's Changed event (discussed in step 2 below) in the constructor.

b. The PrizeAwarded event has object and EventArgs type parameters. When called, this method outputs to the console screen a message informing the user that the prize was awarded to the donor.

c. The method Detach sets the prize to null.

d. Save the library. If you build the library, you will receive two errors because code for the event handler is commented out.

2. Edit Prize:
   a. In the ETS namespace, uncomment the code that declares a delegate event handler with object and EventArgs type parameters.
   b. In the Prize class:
      1. Uncomment the code that declares the event Changed of delegate type.
      2. Uncomment the virtual method OnChangePrize with an EventArgs type parameter. This method will check if Changed is null. If Changed is not null, it passes the current prize instance, and an event argument.
      3. Uncomment the method ClearPrize. It sets the current number of prizes to 0. It will also call OnChangePrize and pass empty EventArgs parameter.
      4. You also need to edit the Decrease method. Uncomment the code that calls OnChangePrize and passes the empty EventArgs parameter immediately following the final line of code to update the prize's current number.
3. Build, save and close the ETS library and open the TelethonSystem client.
   a. Add a new listener instance.
   b. Immediately following the call to AddDonor, add a call to prize's ClearPrize 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 5:

1. The TelethonSystem 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 NewSponsor method by enclosing the code prompting the user for sponsor information and the call to the AddSponsor, and for prize information and the call to AddPrize.
    b. Create a NewDonor method by enclosing the code prompting the user for donor information and the call to the AddDonor, and for donation information and the call to RecordDonation.
    c. 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 both the ETS library and TelethonSystem.
2. Save your work for submission.
3. Keep a copy of the ETS 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 TelethonSystem, add user input validation to ensure numeric values when required.
2. Add one or more of the following methods to the library:

   FindSponsor(string) : string

   This method belongs to the ETSManager class. It receives the sponsor’s ID, and iterates through the sponsors comparing the IDs to the value. If the ID is found, it returns the ID otherwise it displays a message stating the sponsor does not exist.

   Add a prompt in TelethonSystem for an ID to pass into FindSponsor.

   AddValue(double, string) : void

   Create AddValue in Sponsors to locate the sponsor and add the total value of new prizes supplied by an existing sponsor to the sponsor’s TotalPrizeValue.

   Modify TelethonSystem to include the FindSponsor method to allow a sponsor to provide more than one prize. If found, the sponsor’s ID should be added to the new prize information.

   DisplayOneSponsor(string) : string

   This method belongs to the ETSManager class. It receives the sponsor’s ID, and iterates through the sponsors comparing the IDs to the value passed in. If the ID is found, it calls the ToString() method in Sponsors otherwise it displays a message stating the sponsor does not exist.

   Add a prompt in TelethonSystem for an ID to pass into DisplayOneSponsor.

   FindDonor(string) : string

   This method belongs to the ETSManager class. It receives the donor’s ID, and iterates through the donors comparing the IDs to the value. If the ID is found, returns the ID otherwise it displays a message stating the donor does not exist.

   Add a prompt in TelethonSystem for an ID to pass into FindDonor.

   To allow a donor to make multiple donations, edit RecordDonation. Move the call to AddDonor outside of RecordDonation so that it may be called independently if RecordDonation is successful.

   DeleteDonor(string) : string

   This method belongs to the ETSManager class. It receives the donor’s ID, and iterates through the donors comparing the IDs to the value passed in. If the ID is found, it calls the Remove method in Donors otherwise it displays a message stating the donor does not exist.

   Add a prompt in TelethonSystem for an ID to pass into DeleteDonor.

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