Art Gallery

JeongHyun An

DESCRIPTION

Introduction:

The project „Art. Gallery" is a management program for an Art Gallery. The program deals with fundamental components of a gallery that deals with its profit. For example, the management can be broken into several parts: customers, catalogue(works), artists, and the gallery itself.

The functions of the program include as follows:

1. An exhibition at an art gallery is opened. Depending on a budget, it can be decided how much of a space to open within the gallery, for the exhibition. The works cannot be displayed anymore, once they are sold.
2. The catalogue of all works is created. The catalogue is a list of description of all artworks from one exhibition. Each work store related information (title, artist, price). Artworks can be added to the gallery only when there is enough space. Also, for each artwork being sold at the gallery, certain amount of fee should be paid to the gallery, from artist's profit which is equivalent to 40% of the artwork price.
3. Customers can only buy the artwork that is not already sold out, but they can claim those that are not payed yet. Those that are already sold will be marked as unavailable, so they cannot be bought again.

Capability of the program:

1. Creating a gallery
2. Assigning information to a specific artwork
3. Displaying the information of each artwork, and artist
4. Removing customer, artwork or artist data
5. Ending one exhibition
6. Displaying the statistics related to the Gallery: number of artworks, scale of the exhibition
7. Returning input parameter. (Budget, artwork information, customers and payments)

Limitations:

1. The gallery stores artworks in a vector as a catalogue, but there is a capacity for the artworks to be put in the catalogue. Also, the gallery has to run with a certain amount of starting budget, in order to set the capacity (as the gallery in the real world would as well have a finite space)
2. The artwork cannot leave the catalogue even after being sold, because the exhibition has to go on. The availability of the artwork will be checked with '1,' and '0' (When the artwork is available, the availability will be '1').
3. One customer can purchase only one artwork.
4. While a exhibition is going on, no artworks can be added in the middle, because the artworks will be sent to the buyers only after the exhibition is over.
5. Removing a customer does not affect what has already been processed in the gallery

CLASS OVERVIEW

1. Gallery Class:

This is the main class, where all data of artworks, artists, customers will be saves as in vectors. Upon construction of the gallery, the capacity and the budget will be decided. The budget increases as artworks are sold, and also, the capacity is possible to increase during the exhibition as long as certain conditions are satisfied. As the total amount of money to run the exhibition over a period of one exhibition is paid at the beginning, the budget only goes up during the exhibition. If all artworks are sold, the exhibition ends. All the data about the gallery can be printed with one function, which asks user for which set of data to be printed. There are also functions that add data to the vectors included in this class. They are connected to their own classes, which required the following classes to be made as friend classes.

2. Artwork(catalogue) Class:

Every artworks information regarding artist, artwork and the availability indicating whether the artwork was sold or not. After the number of artworks as referred to as capacity in the Gallery class goes through the conversion from budget, the number can be used to determine how many number of artworks can be added. Since there can be the same number to titles, duplicates of the same artwork title is not handled as exceptional case. After purchase, the price of artwork will be converted into profit for both artist and the gallery. 60% for artist, and 40% for gallery.

3. Customer Class:

This is a class for customers. Information about their name, the work they are the most interested in, the date of their visit, and the money they plan to pay at the gallery will be stored. If they run out of money, they cannot buy the artwork. However, if they own money, the money will be automatically withdrawn once they enter the gallery.

4. Artist Class:

This is a class for artists. Since there are overlapping information with artworks class, the artist class takes data from the artwork class. However, these still exist as separate classes, since they are not of the same kind, for example as creature, or object.

5. User Interface Class:

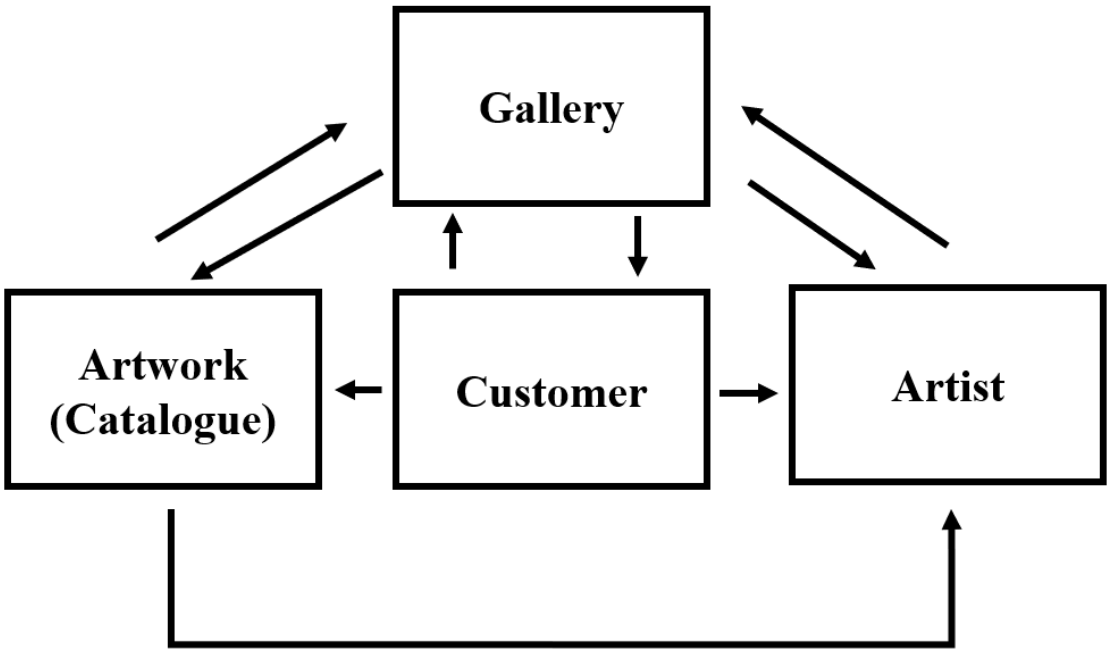
This is a class that contains functions that enable the program to use user inputs from the console. This allows adding elements to the vector, and printing out data.

6. Test Class:

This is a class consisting of test functions which checks the behavior of the program correlated with the artworks input. For instance, it checks if the program correctly prevents adding more artworks than the limit, or whether it is possible to purchase artwork using one of the function.

MEMORY MAP

Fig 1. The figure below is a memory map representing the relations, as outlined in the introduction.



DATA STRUCTURES

Vectors:

```
vector<Artwork> Artworkl;
```

```
vector<Artist> Artistl;
```

```
vector<Customer> Customerl;
```

These vectors store data of artworks, artist, and customers in each. These behave as unique containers throughout the program, which many functions access and modify the data of. The artwork, and customer vectors are create by user inputs, and the artist class is made with data taken from the artwork list. Meanwhile, the customer vectors contain data which may affect the data in artwork and the customer vectors.

DETAILED CLASS:

1. Gallery

```
class Gallery
{
    string galleryName;
    int capacity =0;
    double currBudget;
public:
    vector<Artwork> Artworkl;
    vector<Artist> Artistl;
    vector<Customer> Customerl;

    void printAll(int a);
    string getGalleryName(){return galleryName;}
    int getCapacity(){return capacity;}
    double getCurrBudget(){ return  currBudget;}
    void increaseCapacity();

    int getNumOfArtworks(){
        int num=0;
        vector<Artwork>::iterator it;
        for(it=Artworkl.begin(); it!=Artworkl.end(); ++it) {
            num++;
        }
        return num;
    }
    Gallery(string name, double budget);
    ~Gallery();
    bool isAllSold();

    //Artworks//
    void printAllArtwork();
    void addArtwork(string title1, string name1, double price1);
    void removeArtwork();
    int returnAvailability(string a);
```

```
    //Artists//
    void removeArtist();
    void printAllArtist();

    //Customers//
    void printAllCustomer();
    void addCustomer();//when adding, modify availability, modify profit
    void removeCustomer(string name, string IntWork, double MoneyOwned);

   friend Customer;
   friend Artist;
   friend Artwork;
};
```

→ *EXPLANATION (VARIABLES):*

string galleryName // Name of the gallery

int capacity =0 // Maximum number of artworks, which is initialized to 0 now, but will be modified using calculation in the constructor

double currBudget // Budget the gallery will own throughout the exhibition. As this is a tracker for the current budget, the amount will consistently change. This will be used in increaseCapacity function, which is a gallery method

vector<Artwork> Artworkl // Vector storing artwork class objects

vector<Artist> Artistl // Vector storing artist class objects

vector<Customer> Customerl // Vector storing customer class objects.

→ *EXPLANATION (public methods):*

string getGalleryName(){return galleryName;} // getter for the gallery name

int getCapacity(){return capacity;} // getter for the gallery capacity

double getCurrBudget(){ return currBudget;} // getter for the current budget

void printAll(int a) // prints out all the information on the gallery. A – a user input parameter used as a call for specific print function.

void increaseCapacity() // function that modifies the capacity of the gallery (number of artworks that can be stored)

Gallery(string name, double budget) // Constructor: name – gallery name, budget – gallery

budget

~Gallery() // Destructor

bool isAllSold() // A boolean function that checks the availability of all the artworks using a counter

int getNumOfArtworks() // Getter for the number of artworks

//Artworks//

void printAllArtwork() // prints out the data about all the artwork in the vector

void addArtwork(string title1, string name1, double price1) // function that adds artwork to the vector. Title1 – name of the artwork, Name1 – artist name, price1 – price of the artwork.

void removeArtwork() // removes artwork from the vector

int returnAvailability(string a) // function that returns availability of the artwork. A – name of the artwork.

//Artists//

void removeArtist() // removes artist from the vector

void printAllArtist() // prints out all information about artwork

//Customers//

void printAllCustomer() // prints out all the data from the customer vector

void addCustomer(string name, string IntWork, double MoneyOwned) // a function that adds a customer to the customer vector. This is one of the most important function, since whenever a customer is added, artwork (availability), artist (profit) and the gallery (budget, is all sold) data are all modified at once, and even determined whether the exhibition should end. Name – name of the customer, IntWork – the name of the artwork the customer is interested in, MonyOwned – the money owned by the custome.

void removeCustomer() // removes customer from the vector. This, however does not revert any changes made by the addCustomer function.

```
friend Customer;
friend Artist;
friend Artwork;
```

These are friend classes, which make the functions contained within these accessible by gallery.

2. Artwork

```
class Artwork
{
```

```
    string artistName = "";
    string artworkTitle ="";
    double price =0;
    int availability =1;
public:
    string getArtistNameA(){return artistName;}
    string getArtworkTitleA(){return artworkTitle;}
    int getAvailability(){return availability;}
    void printArtwork();
    double getPrice(){return price;}
    Artwork(string a, string b, double c);
   ~Artwork();
    void makeAvailabilityZ(){this->availability = 0;}

    friend Gallery;

};
```

→ *EXPLANATION (VARIABLES):*

string artistName = "" // name of the artist that is initialized

string artworkTitle ="" // title of the artwork which is initialized

double price =0 // price of the artwork that is initialised to 0

int availability =1 // availability of the artwork that is initialized to 1 which indicates it being available.

→ *EXPLANATION (PUBLIC METHODS):*

Artwork(string a = "No title", string b = "No name", double c = 0.0)//
constructor:

string a - the artwork title

string b – the name of the artist

double c - the price of the artwork

string getArtistNameA(){return artistName;} // getter for the name of the artist

string getArtworkTitleA(){return artworkTitle;} // getter for the title of the artwork

`int getAvailability(){return availability;}` // getter for the availability

`void printArtwork()` // prints out information about a single artwork, used by the printAllArtwork function in the gallery class

`double getPrice(){return price;}` // getter for price

`~Artwork()` // destructor

`void makeAvailabilityZ(){this->availability = 0;}` // used in the customer class, addCustomer function. When the customer is capable of buying a work, the sold out work gets its availability switched to 0 using this function.

`friend Gallery` // friend class: Gallery

3. Artist

```
class Artist
{
    string artistName = "";
    string artworkTitle ="";
    double profit =0;
public:
    Artist(string name="", string title="");
    ~Artist();
    void printArtist();
    void increaseProfit(double val) { this->profit += (0.6*val); }
//used by customer's addTo
    string getArtworkTitle(){return artworkTitle;}
};
```

→ *EXPLANATION (VARIABLES):*

`string artistName = ""` // name of the artist that is initialized

`string artworkTitle =""` // title of the artwork which is initialized

`double profit =0` // profit of the artist that is initialised to 0, which will be modified when artworks get sold

`int availability =1` // availability of the artwork that is initialized to 1 which indicates it being available.

→ *EXPLANATION (PUBLIC METHODS):*

Artist(string name="", string title="") //

constructor:

string a - the artwork title

string b – the name of the artist

~Artist() // destructor

void printArtist() // prints out information about a single artist, used by the printAllArtwork function in the gallery class

void increaseProfit(double val) { this->profit += (0.6*val); } // used by customer Class' addCustomer function. This modifies the profit variable.

string getArtworkTitle(){return artworkTitle;} // getter for the artwork title

4. Customer

```
class Customer
{
private:
    string customerName;
    string interestedWork;
    double moneyOwned;

public:

    Customer(string a =" no name", string b ="Work", double c = 0);
    void printCustomer();
    string getCustomerName(){return customerName;}
    string getInterestedWork(){return interestedWork;}
    ~Customer();

    friend Gallery;
};
```

→ *EXPLANATION (VARIABLES):*

`string customerName //` name of the customer that is initialized

`string interestedWork //` title of the artwork which is the customer is interested in

`double moneyowned //` money owned by the customer

→ *EXPLANATION (PUBLIC METHODS):*

`Customer(string a =" no name", string b ="Work", double c = 0) //`

constructor:

`string a -` customer name

`string b –` interested work

`double c -` moneyowned

`void printCustomer()//` prints out information about a single customer, used by the printAllArtwork function in the gallery class

`string getCustomerName(){return customerName;} //` getter for a customer name

`string getInterestedWork(){return interestedWork;} //` getter for an interested work

`~Customer() //` destructor

`friend Gallery //` friend class: Gallery

5. Test

```
class Test{
public:
    int test1();
    int test2();
    int test3();
};
```

→ *EXPLANATION (PUBLIC METHODS):*

`int test1()//` tests if the program prevents user from adding more artwork elements than the capacity (limit

on the number of artworks)

int test2()// this is a sub-test of the test 1. Created in order to complement the test 1, and shows that the number of the artwork input can only be less than or equal to the value of the capacity. Otherwise, they won't be stored in the vector.

int test3()// tests if the addArtwork(), and addCustomer() both work correctly so that addArtwork() does store elements in its vector, and addCustomer() modifies the elements within the Artwork vector.

5. User Interface

```
class UserInterface{
public:
    void addCustomer(Gallery &a);
    void addArtwork(Gallery &a);
    void printAll(Gallery &a);
};
```

→ *EXPLANATION (PUBLIC METHODS):*

void addCustomer(Gallery &a)// adds customer to the vector through user interface. This function asks to user on the console about the details of the element one by one.

void addArtwork(Gallery &a)// adds artwork to the vector through user interface. This function asks to user on the console about the details of the element one by one.

void printAll(Gallery &a)// shows user the options for each print function.

Changes made:

---

1. **Adding/removing artworks**
    a. I originally aimed to remove all the artworks from the gallery once the exhibition is over, however throughout the coding stage, I realized that destructor performs this

job, and the removal is completely unnecessary. Thus I decided to print out all the artworks with their sold out details at the end instead.

b. I also thought artworks could not be removed or added while exhibition is going on, but later, I wanted to make this more flexible to prepare for wider variety of situations, and made this available by adding the remove and add artworks function.

2. Period

a. The original idea I had was to end a exhibition once period is exceeded, which could be handled with each customer's visit date. This idea later changed to having setting maximum number of customers per one exhibition. However, I did not want to set limitation on the user input, and wanted to make the story more natural, and therefore I changed this to ending exhibition when all artworks are sold.

3. Capacity

a. I wanted to make the capacity calculation include dimension of artworks, but this did not seem very applicable to real life, so I decided to only take the number of artworks instead, assuming all artworks will have enough space distributed.

4. Usage of vectors

a. At first, I planned to use doubly linked list with pointer, but as I wanted to achieve more flexibility I decided to use vector instead.

5. Relocating the functions

a. Because I changed the containers from linked list to vectors, where all the vectors are deeply correlated with each other, I moved the vector lists to the main class, and put the related methods in each of the classes which related to the vectors.

6. Separating user interface from the functions that add elements to the vectors.

a. Before, I had all the user interfaces and the inner work of the program, so that each function would directly obtain inputs from the console. However, this made the functions very complex, and made it unable to work flexibly as such that it was impossible to push back an existing element to the vector. Thus, I separated this by moving all interfaces work to a new class called userInterface class.

7. Adding a test class

a. Since the code lacked interactive testing part, I decided to put another separate test class where I add functions that performs various tests, and shows how exactly the specific parts of the program were working.

Test:

---

*There are not many lines of codes in the main function, as I tried to finish the implementation for all user interactions in the class methods.

```
int main()
{
    Test t = Test();

    int errcode1 = t.test1();
    int errcode2 = t.test2();
    int errcode3 = t.test3();
    cout<<"\n\n";
    int num;
    if(errcode1!=0){
        num++;
        cout<<"Error code: "<<errcode1<<endl;
    }
    if(errcode2!=0){
        num++;
        cout<<"Error code: "<<errcode2<<endl;
    }
    if(errcode3!=0){
        num++;
        cout<<"Error code: "<<errcode3<<endl;
    }
    cout<<"Number of tests passed: "<<num<<endl;

    cout<<"Testing User Interface"<<endl;
    Gallery b("ArtGallery", 700);
    b.addArtwork("title1", "artist1", 50);
    b.addArtwork("title2", "artist2", 70);

////Here I use user interface.
    userInterface u;
    u.addArtwork(b);
    u.addCustomer(b);
```

```
    u.addCustomer(b);
    u.printAll(b);
    b.removeCustomer();
    b.removeArtwork();
    return 0;
}
```

[Console]

```
"C:\Users\jeong\CLionProjects\Art Gallery\cmake-build-debug\untitled1.exe"

//////////Testing if the program prevents number of artworks exceeding the
capacity//////////

Creating a Gallery of name: Test1Gallery
Budget: 0
Number of Artworks: 3
!!!Test Passed!!!
Number of artworks: 3
Artwork title: title1
Artist name: artist1
Price of artwork: 50
Artwork state: Available
Artwork title: title2
Artist name: artist2
Price of artwork: 60
Artwork state: Available
Artwork title: title3
Artist name: artist3
Price of artwork: 70
Artwork state: Available

//////////Testing if the program prevents number of artworks exceeding the
capacity//////////

Creating a Gallery of name: Test2Gallery
Budget: 0
Number of Artworks: 3
```

```
!!!Test failed!!!


//////////Testing if it is possible to purchase an artwork//////////

Creating a Gallery of name: Test3Gallery
Budget: 0
Number of Artworks: 3
!!!Test Passed!!!
Artwork title: title1
Artist name: artist1
Price of artwork: 60
Artwork state: Available
Artwork title: title2
Artist name: artist2
Price of artwork: 60
Artwork state: Sold
Artwork title: title3
Artist name: artist3
Price of artwork: 70
Artwork state: Available
Customer purchased one artwork


Error code: 2
Error code: 10
Number of tests passed: 2
Testing User Interface
Creating a Gallery of name: ArtGallery
Budget: 0
Number of Artworks: 5
Enter title:title3
 title3
Enter name:artist3
 artist3
Enter price :40
 40
Enter customer name:customer1
 customer1
Enter customer Interested work:title2
 title2
```

```
Enter Money Owned:80
 80
Enter customer name:customer2
 customer2
Enter customer Interested work:title1
 title1
Enter Money Owned:10
 10
---------INFORMATION ON GALLERY-------------
Name of the Gallery: ArtGallery
Budget: 48
Number of artworks: 5
If you want to view the information on artists, enter 1
For the information on artworks, enter 2
For the information on customers, enter 3
2
2
Artwork title: title1
Artist name: artist1
Price of artwork: 50
Artwork state: Sold
Artwork title: title2
Artist name: artist2
Price of artwork: 70
Artwork state: Sold
Artwork title: title3
Artist name: artist3
Price of artwork: 40
Artwork state: Available
Who would you like to remove?title3
title3
What would you like to remove?title3
 title3
```