CS-224 Object Oriented Programming and Design Methodologies

Homework 04

Spring 2022

1 Guidelines

You need to submit this homework on 18th March at 8pm, on LMS. Late submissions are allowed until 20th March 11:59pm, which will be penalized by 20%. Your work will not be accepted once the submission is closed on LMS.

- You need to do this assignment in a group of two students.
- You will submit your assignment to LMS (only one member of the group will submit).
- Clearly mention the group composition in submitted file name e.g. AhmadHassan_ah01345_BatoolAiman_ba03451.zip.
- You need to follow the best programming practices
- Submit assignment on time; late submissions will not be accepted.
- Some assignments will require you to submit multiple files. Always Zip and send them.
- It is better to submit incomplete assignment than none at all.
- It is better to submit the work that you have done yourself than what you have plagiarized.
- It is strongly advised that you start working on the assignment the day you get it. Assignments WILL take time.

- DO NOT send your assignment to your instructor, if you do, your assignment will get ZERO for not following clear instructions.
- You can be called in for Viva for any assignment that you submit

2 Bank Accounts

In this question you are going to develop a little simulation of banking accounts. You need to think in terms of objects encapsulated with attributes and their functions.

The first object is a `Bank` which have a `vector` of `Accounts`. Each Account object have certain balance and it can be manipulated with a `Deposit` and `Withdrawal` transactions. To maintain record, the `Account` need to store all the transactions executed on it, hence vectors of transactions can be part of an `Account` object. After executing all the transactions any account with less than 5000 available balance will become dormant, otherwise it is active. A `Date` class can be created that handles all tasks related to it.

You need to model all the classes can be found in this scenario e.g. `Bank, Account, Deposit, Withdraw, Date...` You may create any other class/-functions/attributes as deemed appropriate or necessary.

Input File Processing

A sample input file is as follows:

```
Create Jasmin_Banda JB230 70000
Deposit GH090 30000 11-Mar
Withdrawal GH090 4000 12-April-21
```

Please refer to file `bankinput.txt` which contains certain entries to process. You can find three different types of entries in the file:

- `Create Title Code InitialDeposit:` This entry creates an account with account title, code and initial deposit
- `Deposit Code Amount Date:` This entry deposits the given amount in the particular account as per code. Date can vary in different forms, you have to adjust all of the forms e.g. default year is 2021, and it should accept all forms of the months.
- `Withdrawal Code Amount Date:` This entry withdraws the amount from given account successfully if the available balance is sufficient to

withdraw the money, else the transaction should fail. Date can vary in different forms, you have to adjust all of the forms e.g. default year is 2021, and it should accept all forms of the months.

Output File

After executing all the transactions from input file, your program should generate an output `bankoutput.txt` file, as per following sample it should print information of all accounts.

```
Account Title: Saeed Ghani
Account Code: GH090
Initial Balance: 2000
Available Balance: 28000
Current Status: Active
Deposits:
1. 30000 on 11/03/21
Withdrawals:
1. 4000 on 12/04/21 Successful
2. 40000 on 15/06/21 Denied
```

3 HUMania

A sample code is given in HUMania folder, if you run it you can see a pigeon is drawn. This example creates just one object of Pigeon to show how things are drawn in SDL. Refer to `Pigeon.hpp/cpp and HUMania.cpp` ⇒ `drawObjects()`.

You are required to:

- Create a `Pigeon` class (see the pigeon.hpp/cpp), that will contain attributes and functions (`fly, draw`) related to a pigeon. The `fly` function flies the pigeon gradually to right side, and rotates through the screen.

- Create a `Butterfly` class (create butterfly.cpp/hpp files), that will contain attributes and functions (`fly, draw`) related to butterfly. The fly function should take the butterfly right-down direction. Once a butterfly reaches to bottom of the screen, it starts flying right-up direction. Once it reaches top of the screen it moves right-down. Similar to the pigeon, it should rotate through the screen.

- Create a `Bee` class (Create bee.cpp/hpp files), that will contain attributes and functions (`fly, draw`) related to a bee. The fly function should make it fly towards right only. During fly it should hover (doesn't move forward) for a while over a random interval. You may choose 1% probability in every frame to decide whether it starts hover, and it keeps hovering for 10 frames. As a bee reaches to right most border of screen, it exits from the game, hence the object must be removed from the bees vector.

- Every object animates three of the images provided in the assets file. The draw function is only drawing the object.

- As you click on the screen, one of the above objects is created randomly. You'll maintain three vectors (pigeons, butterflies, bees) in `HUMania.hpp/cpp` to store objects of different classes. The object that you create on the click will be pushed into corresponding vector. Refer to `HUMania.cpp` ⇒ `createObject()`, where you get mouse coordinates.

- You have to create objects dynamically with `new` operator, hence the vectors should hold pointers to all of the objects. Remember to delete the objects when game is ended, and when the bee objects are removed from vector.

- Finally, you iterate over all the elements of vectors, and call their fly and draw functions.

- Please refer to `Solution.exe` file to see it all in action.

- Are you having fun?? You are more than welcome to add more stuff to make this game interesting, e.g. some natural random movement of butterflies, sitting them on ground, pigeons sweeping etc. [*It doesn't carry any marks*]

3.1 SDL Drawing Basics

The basic drawing function in SDL is very simple, you need two SDL_Rect variables to draw a portion of image from assets file to the canvas. `SDL_Rect` is a simple structure containing {`x, y, w, h`} attributes. `(x, y)` is the top-left corner, and `w, h` are width and height of rectangle. You define a `srcRect` for desired object in assets file, and define a `moverRect` for this

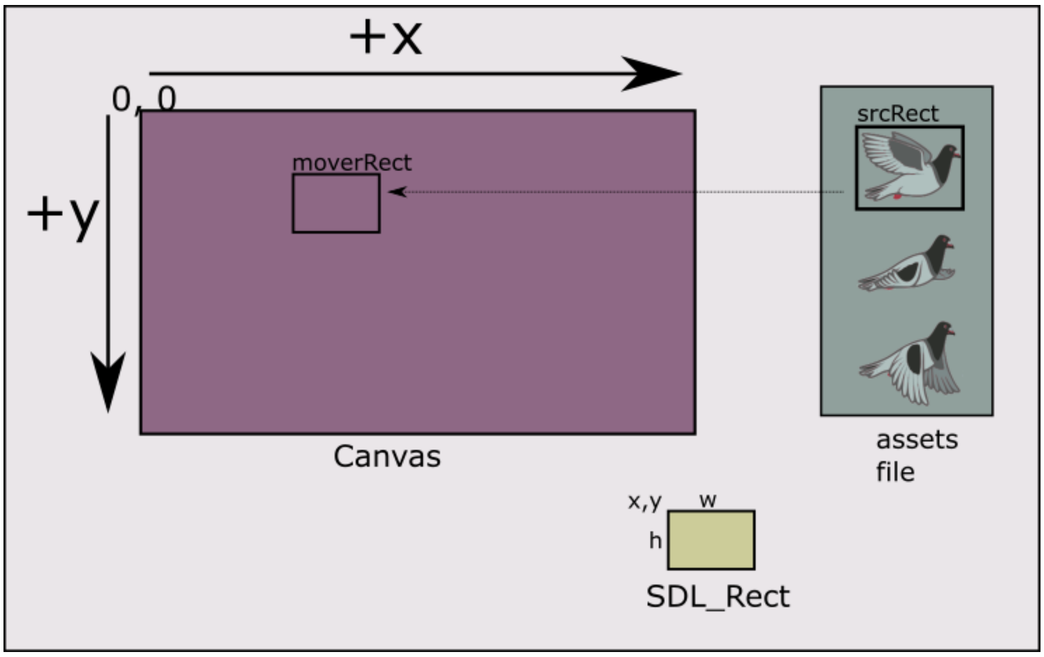

Figure 1: SDL Drawing Basics

image to be drawn on desired location on canvas. Refer to Figure 1 for all this process. Finally you call
`SDL_RenderCopy(gRenderer, assets, &pigeonSrc, &pigeonMover);`
that displays this image to the canvas, voila!!!. Refer to `assets.png` file for all the required image assets.

You can draw as many objects in the `HUMania.cpp` ⇒ `drawObjects()`, as you want. Since this function is called infinitely, you can change the `x, y` attributes of `moverRect` to move the objects on screen, and you can change the `srcRect` values to get a flying animation.

4 `std::vector` Tutorial

Following is a basic example to work with vector. Complete reference for C++ vector is given here `https://en.cppreference.com/w/cpp/container/vector`

```
#include<iostream>
#include<vector>

using namespace std;
```

```
class Distance{
   int feet, inches;
   public:
   Distance(int ft, int inch): feet(ft), inches(inch){}
   void show(){
      cout<<feet<<"'"<<inches<<"\""<<endl;
   }
};

int main(){
   vector<Distance*> dst; // It's a vector that can store Distance
       type objects
   dst.push_back(new Distance(3, 4)); // create an object, and push
       it in vector
   dst.push_back(new Distance(5, 2));
   dst.push_back(new Distance(2, 7));
   dst.push_back(new Distance(7, 8));
   dst.push_back(new Distance(13, 1));

   for(int i=0;i<dst.size();i++)
      dst[i]->show(); // call show method of dst[i] object

   // deleting the objects, need to delete every single object
       created dynamically
   for(int i=0;i<dst.size(); i++)
      delete dst[i];

   dst.clear(); //clears all the items from vector

}

//////////////// Output: ///////////////////
3'4"
5'2"
2'7"
7'8"
13'1"
```

5 Some important points:

- Sample code is there for your benefit. If you are going to use it, understand how it works.
- You do not need to follow the code given exactly. You can make changes where you see fit provided that it makes sense.
- Make the class declarations in hpp files, and provide function implementations in cpp files. Don't use hpp files for implementation purposes.
- A tutorial given here to remove the elements from vector, you might need it to remove bees as they exit the screen.
- As a general rule, class's deta is private, and functions are public. Don't use getter/setter functions to manipulate data, rather think in object oriented directions and provide all the functionality in the class.
- Complete reference for C++ vector is given here `https://en.cppreference.com/w/cpp/container/vector`
- You need to define separate `*.hpp` and `*.cpp` files for all the classes.
- Exact x,y,w,h values for images in assets file can be found by `http://www.spritecow.com/`.
- A tutorial for file I/O is given `http://www.cplusplus.com/doc/tutorial/files/`.
- You should take `www.cplusplus.com` and `www.cppreference.com` as primary web source to search about C++
- You have to follow best OOP practices as discussed in lectures.

6 How to compile

Open the given `Seeplusia` folder in vscode by choosing `File` $\Rightarrow$ `Open Folder`. The game can be run by simply pressing F5 from vscode. If due to some reason it doesn't work, then go compiling and running it from terminal, as explained in `how to compile.txt`

7 Rubric

Coding	The code followed best practices guideline	1
OOP Concepts	The code followed best OOP practices	2
Memory	Dynamic memory management is done properly	2
Functionality	All the functionality is implemented as described above	5
Total		10

Table 1: Grading Rubric