Lab Manual – Composition and Aggregation

Objectives

After completing this lab, you will be able to:

- Identify and implement the “Composition” relationship between classes
- Identify and implement the “Aggregation” relationship between classes

Definitions of Aggregation and Composition

Aggregation:
The aggregation relationship is used to represent the ownership or a whole/part relationship between classes. The aggregate object has one or more parts which may be shared with other objects of the same class or other classes. The objects that make up the parts are created and destroyed independently of the aggregate object.

Composition:
Composition is used to represent a stronger kind of ownership than the aggregation relationship. In composition, the composed object has full responsibility for the disposition of its parts in terms of creation and destruction. Talking in terms of implementation, the composite has the responsibility for memory allocation and de-allocation of its parts. Moreover, the parts of a composite object cannot be shared with other objects.

Composition

Exercise 1:

Make a new application called `Lab_<your roll number>`. Define and implement a class `Point` in files `Point.h` and `Point.cpp`, respectively. This class should provide:

- Two private integer data members `x` and `y` which will store the x and y coordinates of a point
- A default constructor which takes two parameters to initialize the x and y coordinates and prints “`Point`() called” on the screen.
- A function `print()` which prints out the point on the screen in the format `(x,y)`
- A destructor which prints “~`Point`() called” on the screen.

Exercise 2:

Now define and implement a class `Circle` in files `Circle.h` and `Circle.cpp`. This class should contain:

```
Class Circle{
     Point center;
     Float radius;
};
```

- A private data member `center` which will be an instance of the `Point` class
- A private float data member `radius` that will store the radius of the circle
- A constructor which takes three parameters (x and y coordinates of the center of the circle, and the radius) and initializes the data members accordingly and also prints “`Circle()` `called`” on the screen.
- A destructor which prints “~`Circle()` `called`” on the screen.
- A function `print()` which prints the information (center and radius) of the circle on the screen

To call the constructor of class `Point` from the constructor of class `Circle`, you can use the following syntax.

```
Circle::Circle(int x, int y, float r): center(x,y) { … };
```

Add another file `Lab.cpp` in your project. Copy the following piece of code in that file, compile and then execute. Note down the output of the program and write it in comments in the code.

```
#include "Circle.h"

void main()
{
      Circle c (3,4,2.5);
      c.print();
}
```

Exercise 3:

Define and implement a class `Triangle` in files `Triangle.h` and `Triangle.cpp`. This class should provide:

- Three private data members x, y and z (`Point` type) which will be indicating the three corners of the triangle.
- A constructor which takes six parameters (x and y coordinates of the three corners) and initializes the data members accordingly and prints “`Triangle()` `called`” on the screen.
- A destructor which prints “~`Triangle()` `called`” on the screen.
- A function `print()` which prints out the information (i.e. the coordinates of its three corners) of the triangle object on the screen

Exercise 4:

Modify the `Lab.cpp` file to instantiate an object of class `Triangle` called obj with parameters for points (1, 0) (0, 1) and (0, 0) and call its `print` function. Note down the output of the program and write it in comments in the code.

(Practice Problem – Not Included in Lab, implement it as Composition)

Exercise 5:

Define and implement a class `Style` in files `Style.h` and `Style.cpp`. This class should include:

- A private data member `char color[10]` which stores the color of the object
- A private boolean data member `isFilled`
- Setters for these data members (i.e. `SetColor` and `SetFilled`)
- A constructor which takes the color and a boolean value and initializes the data members. The constructor should also print "`Style() called`" on the screen.
- A destructor which prints "`~Style() called`" on the screen.
- A function `print()` which prints the data members of the style on the screen

Exercise 6:

Now modify the `Circle` class to include a `Style` called `st`. Note that you will have to modify the constructor and `print` function of `Circle` class accordingly. Initially, a newly constructed `Circle` will have some default style. At this stage, you will have to add three more methods to the Circle class:

- Method `SetStyle (Style&)` will take a Style and set `st` to the object passed as an argument
- Method `SetColor (char[])` will update the color of the circle. You will call the `SetColor` method of `Style` class inside this method.
- Method `SetFilled (bool)` will update the filled property of the circle. You will call the `SetFilled` method of `Style` class inside this method.

Update Styles of Circle and Triangle and test this functionality in main.

Aggregation (Practice Problem – Not Included in Lab)

Exercise 8:

Define and implement a class `CompactDisc` in files `CompactDisc.h` and `CompactDisc.cpp`. This class should include:

- A private data member `char title[20]` indicating the title of the CD.
- A private integer data member `capacity` that stores the capacity of the CD in MBs
- A constructor which takes the title and capacity of the CD to initialize these data members. It should also print "`CompactDisc() called`" on the screen.
- Getters and Setters for title and capacity of the CD.
- A destructor which prints "`~CompactDisc() called`" on the screen

Exercise 9:

Define and implement a class `CDDrive` in files `CDDrive.h` and `CDDrive.cpp`. This class should include:

- A private data member `char manufacturer[20]` indicating the name of the manufacturer of the CD Drive
- A private integer data member `speed` which stores the read/write speed of the CD Drive
- A private `CompactDisc` pointer named `aCompactDisc` which points to the currently inserted Compact Disc.
- A constructor which takes the manufacturer and speed of the CD Drive to initialize the members. Please note that on initialization, the CD Drive will not have any CD inserted in it so the pointer should point to NULL on initialization. It should also print "`CDDrive() called`" on the screen.
- A public method called `InsertCD` which takes an address of a `CompactDisc` object and sets the data member `aCompactDisc` to this address.
- A public method called Play which does the following steps
    - Get the title and capacity of the Compact Disc and displays it
    - If no CD was present inside the CD Drive it should display "`Please insert a disc.`"
- A public method called `EjectCD` which takes no argument and returns the pointer to the CD that was present inside the CD Drive and resets the private data member `aCompactDisc`. If no CD was present, a null pointer should be returned.
- A destructor which prints "~`CDDrive`() called" on the screen

Exercise 10:

Write a driver for the above classes which does the following and note down the out put of the program and write it in comments.

- Instantiates two objects of class `CompactDisc` called `cd1` and `cd2`.
- Instantiate an object of class `CDDrive` called `theCDDrive`
- Insert `cd1` in `theCDDrive`
- Play `theCDDrive`
- Eject `theCDDrive`
- Play `theCDDrive`
- Insert `cd1` in `theCDDrive`
- Play `theCDDrive`
- Eject `theCDDrive`