Object-Oriented Concepts and Programming

Lab 14

Start your program with your information using the following format:

```
/*
ID:
Name:
Lab No:
Question No:
Date:
*/
```

1. Apply the proper design patterns to customer mail application program in Lab 11 so that the CustomerMailApplication class can use only Customer class.
2. The following C++ class is a simulation of a ScheduleServer class, it contains functions to start, initialize, shutdown and so on.

```
class ScheduleServer {
  public:
    void startBooting(){
            cout << "Starts booting..." << endl;
    }
    void readSystemConfigFile(){
       cout << "Reading system config files..." << endl;
    }
    void init(){
            cout << "Initializing..." << endl;
    }
    void initializeContext(){
            cout << "Initializing context..." << endl;
    }
    void initializeListeners(){
```

```
            cout << "Initializing listeners..." << endl;
    }
    void createSystemObjects(){
       cout << "Creating system objects..." << endl;
    }
    void releaseProcesses(){
       cout << "Releasing processes..." << endl;
    }
    void destory(){
       cout << "Destorying..." << endl;
    }
    void destroySystemObjects(){
            cout << "Destroying system objects..." << endl;
    }
    void destoryListeners(){
       cout << "Destroying listeners..." << endl;
    }
    void destoryContext(){
       cout << "Destroying context..." << endl;
    }
    void shutdown(){
       cout << "Shutting down..." << endl;
    }
};
```

The main program was written as follows to start and shutdown the server.

```
int main() {
   ScheduleServer scheduleServer;
         scheduleServer.startBooting();
```

```
        scheduleServer.readSystemConfigFile();
        scheduleServer.init();
        scheduleServer.initializeContext();
        scheduleServer.initializeListeners();
        scheduleServer.createSystemObjects();
        cout << "Start working....." << endl;
        cout << "After work done....." << endl;
        scheduleServer.releaseProcesses();
        scheduleServer.destory();
        scheduleServer.destroySystemObjects();
        scheduleServer.destoryListeners();
        scheduleServer.destoryContext();
        scheduleServer.shutdown();
        return 0;
}
```

The point is the main program was not well organized, since it has too many statements, actually the main tasks can be divided into only two parts start the server (all processed before the line cout << "Start working....." << endl) and shutdown the server (all processes after the line cout << "After work done....." << endl) Your task is to apply the proper design pattern(s) to organize the main program to call only functions to start and to stop server.