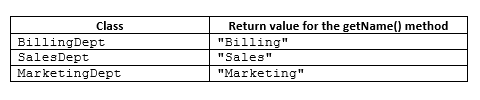
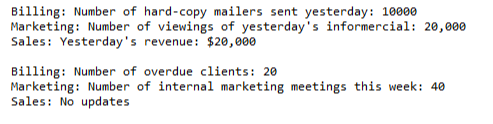
Problem Statement. [UML to JAVA Code] In a company, the administrative office receives hourly formatted reports from the Billing, Sales, and Marketing Departments. Each of these departments provides a message queue, which the administrative office reads each hour. When it is time to read the department queues, the administrative office software reads each queue's message and then assembles all the messages into a report. A typical formatted report looks like this:



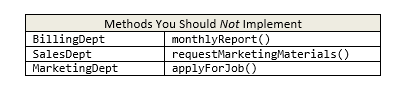
For this problem, you will implement classes BillingDept, SalesDept, MarketingDept, and their superclass Department. Note from the class diagram (below) that each of these subclasses of Department implements the method getName(). In the table below, the return values of this method are provided:



Class Return value for the getName() method BillingDept "Billing" SalesDept "Sales" MarketingDept "Marketing" You will also implement an Admin class and the method hourlyCompanyMessage(), which reads the message in each of the Department queues and assembles them into a report, returned as a String. In order to assemble the messages and organize them into the correct format, the format() method in Admin must be called. The Department queues are implementations of a special queue class that has been provided for you, called StringQueue. The StringQueue stores messages within each department class, and your hourlyCompanyMessage() implementation will read each of the departmental queues to get the current message from each. It is possible that, when you access one of the Departmental queues, an EmptyQueueException (which is a class that has been implemented for you) could be thrown; your hourlyCompanyMessage() method must handle any such thrown exception. There is a test class, Main, whose main method provides test data to test your code. The expected output of the main method (after commented sections have been uncommented) is:



Important: Each of the departments BillingDept, SalesDept, MarketingDept has a method besides getName(), indicated in the class diagram below. These methods have already been declared for you – you should not implement these methods. The table below lists these methods and the class each belongs to; remember, these methods do not need to be implemented and they have already been declared for you.



Methods You Should Not Implement BillingDept monthlyReport() SalesDept requestMarketingMaterials() MarketingDept applyForJob()

Note: And, you are strongly advised to avoid modifying any part of the Main class, except for uncommenting the commented part of the code when you are ready to use it .

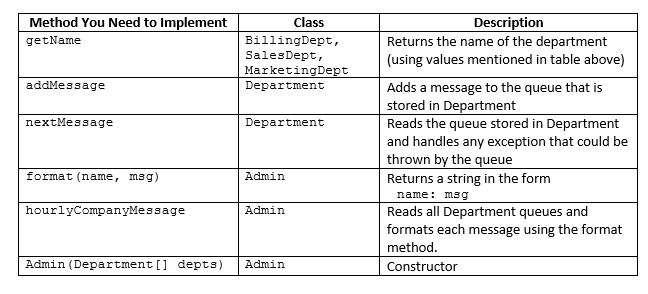
Note: The Main class plays the role of an actor in this piece of software, and is indicated this way in the sequence diagram. (For this reason, Main does not appear in the class diagram.)

Tasks.

(1) Carefully implement the classes shown in the class diagram, with behavior shown in the sequence diagram (below), observing multiplicities, roles, and stereotypes.

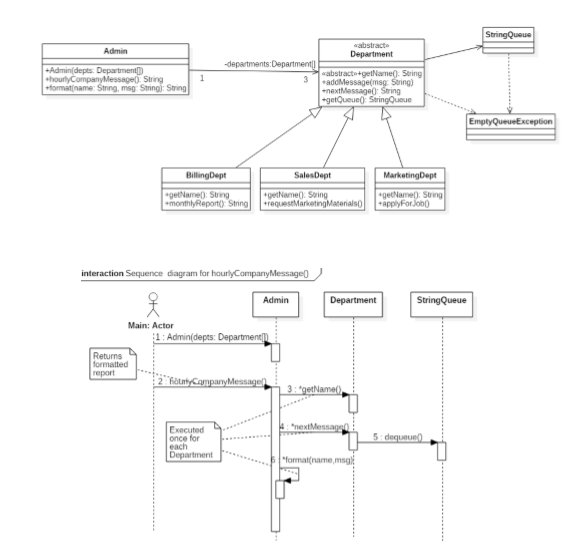
(2) Implement all operations shown in the class diagram, except for those in the table above.

(3) The most important implementation you need to do is for the Admin method hourlyCompanyMessage. During evaluation of your code, the expected output of this method (shown above) will be compared with yours. To test your output, use the main method provided for you in the Main class.



Requirements for this problem.

(1) You must use the StringQueue class provided whenever a queue is needed in your code. (2) Your implementation of the Admin method hourlyCompanyMessage must use polymorphism to read messages; you are not allowed to check types (using instanceof or getClass()). (3) The output of the Admin method hourlyCompanyMessage must be formatted as it has been done in the samples shown above. (4) All operations shown in the class diagram must be implemented, except for the operations monthlyReport(), requestMarketingMaterials(), applyForJob(). (5) Your hourlyCompanyMessage must call the format() method to perform the necessary formatting of messages. (6) The flow of your code, when hourlyCompanyMessage is executed, must match the sequence diagram shown below. (7) The nextMessage() method in Department must read the next value in the StringQueue and return it. Since reading the StringQueue could cause an EmptyQueueException to be thrown, you must make use of a try/catch block. The body of the catch block can be left empty (you do not need to handle an EmptyQueueException in any special way). (8) The String returned from hourlyCompanyMessage() in Admin must have the following format (which is produced by the format(name, msg) method):

 <department\_name>: <message> For example: Billing: This is a message from the BillingDepartment The department name that appears in the output is obtained by calling the getName() method. (9) You must not modify the code in StringQueue or EmptyQueueException. (Note that these two classes are shown in the diagrams, but implementation details are not shown since they are already fully implemented.) Your submitted code must not have compiler errors or runtime exceptions when executed.