Student ID: _____

Full Names: _____

Modern Programming Practices (CS401)

(February 2021)

Teacher: O. Kalu

Midterm Integration (Part 2)

1. The exam (both parts 1 and 2) duration is 2 hours.
2. This part 2 of the exam is computer-based; so you should use a computer for the tasks.
3. **If you are taking the Exam remotely online, you are required to keep your computer camera on and stay visible at all times during the exam session. Failure to do so, will cancel and disqualify your Exam**.
4. **This exam is a copyrighted material and must not be taken away, or copied or photographed or reproduced or transferred or shared or distributed**.
5. You are expected to use an IDE or any Code Editor tool of your choice to implement your solutions for the questions in this part.
6. Upon completion, make sure to put your entire work, **(including your design diagram(s), screenshots, entire project(s) with the source code etc.)** into a single compressed (e.g. zip) file named **MidtermExamPart2.zip**, and submit to Sakai, under the Assignment titled, "MidtermExam".
7. When you **take** your **screenshot**s, it should be **of the entire computer screen** (NOT a snippet or a window – see an example screenshot below, at the end of this document)
8. NO CHEATING!!!

Good luck.

(CS401 – MPP)
(February 2021)
Midterm Examination – Part 2 - Coding (60 points)

Part II – OOAD, Coding and Problem-solving skills: (60 points)

Note: *One of the core areas of the MPP curriculum is implementation of a software design model, from UML to Java code. Given below, are two Object-oriented analysis/design, coding/problem-solving questions that will test your skill level/competency in this area. For each of these questions, you are expected to take a screenshot of your work in the computer showing your code in the IDE Window and the console output/result, save it as an image file (using png or jpg format) and include it in a "screenshots" folder/directory inside your project folder, which you will zip into the one/single zip file, which you submit to Sakai.*

Also, when you take the screenshots, it should be of the entire computer screen (NOT a snippet or a window – see an example screenshot below, at the end of this document)

1. (20 points) **Draw UML Static model and implement Java code for an SRM App**

Assume you have been hired by a local supermarket named, MartWal, to design and implement a Supplier Relationship Management (SRM) system, which they will be using to manage the inventory (list) of Products that they stock, along with the various Suppliers who do supply them with the products.

Each Product is supplied by a specific Supplier. And, **a Supplier supplies one or more Products** to the supermarket. The supermarket assigns a unique Product Number to every product that they stock and they are also interested in maintaining the following pieces of data about each product: the product name, its unit price, quantity in stock and the date it was supplied.

Each Supplier who supplies one or more products to the supermarket is given a unique supplier identification number and the supermarket also needs to maintain data about the supplier's name.

Here is the data specification for the system, showing the two domain model classes and their respective attributes:

Supplier: supplierId, supplierName
Product: productNumber, productName, unitPrice, quantityInStock, dateSupplied

For this question, you are required to do the following tasks:

a. Draw a UML Static (Class) model for the system.

*Make sure to include in your diagram, the correct relationship between Supplier and Product, and showing the correct multiplicities; and also indicating such that it is **bidirectional**. (see the sample data presented in Table 1, below).*

b. Create a Java Command-line (Console) application project and implement code for the solution model. In your code of the model classes, make sure to include appropriate constructor(s) and the getter (accessor)/setter (mutator) methods for each of the data fields (properties). Make the domain model classes be inside a package named, '*edu.miu.cs.cs401.midtermp2.srmapp.model*'.

c. Code an executable class named, SRMApp, and in it, include the main(…) method. And, in the main(…) method, add code to create an **array** of Suppliers using the following data:

MartWal Supermarket's Suppliers-and-Products data for the SRM system						
SupplierId	**SupplierName**	**ProductNumber**	**ProductName**	**UnitPrice**	**QuantityInStock**	**DateSupplied**
S101	United Farms	3109128478	Bananas	$ 1.25	480	2021-02-14
S101	United Farms	2910019138	Apples	$ 1.09	525	2021-01-31
S102	ElSegundo Agro	7281100287	Avocados	$ 2.49	164	2021-02-11
S103	La Boulangerie	2102799156	Brioche	$ 1.89	96	2021-01-04

Table 1: MartWal Supermarket's Suppliers-Products data *(Note that: ProductNumber is numeric NOT alphanumeric or otherwise)*

Make the SRMApp class be inside a package named, '*edu.miu.cs.cs401.midtermp2.srmapp*'.

In the main(…) method, also add code to iterate through the array of Supplier and print-out to the console, each Supplier data together with its respective Product(s) data.

2. (40 points) **Company Administration system - Translate UML to Java Code**

Note: *For this question, you will use the Java classes that have been provided for you inside the Starter-Code package zip file named*, "***Starter-Code-for-MidtermExamPart2-Q2.zip***", *which you will download from the Sakai Assignment titled, "**MidtermExam**". You will complete the necessary coding in these classes, following the instructions provided below.*

You will use both the class diagram and Sequence diagram given at the end of the question, to implement your solution.

Caution! Please make sure to carefully read the entire problem statement given below and pay attention to every detail, before beginning implementation of your solution.

Problem Statement:

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:

```
Billing: Number of hard-copy mailers sent yesterday: 10000
Marketing: Number of viewings of yesterday's informercial: 20,000
Sales: Yesterday's revenue: $20,000
```

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:

```
Billing: Number of hard-copy mailers sent yesterday: 10000
Marketing: Number of viewings of yesterday's informercial: 20,000
Sales: Yesterday's revenue: $20,000

Billing: Number of overdue clients: 20
Marketing: Number of internal marketing meetings this week: 40
Sales: No updates
```

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

For this question, here are the tasks you need to accomplish:

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* class's 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.

Method You Need to Implement	**Class**	**Description**
getName	BillingDept, SalesDept, MarketingDept	Returns the name of the department (using values mentioned in table above)
addMessage	Department	Adds a message to the queue that is stored in Department
nextMessage	Department	Reads the queue stored in Department and handles any exception that could be thrown by the queue
format(name, msg)	Admin	Returns a string in the form *name*: *msg*
hourlyCompanyMessage	Admin	Reads all Department queues and formats each message using the format method.

Requirements for this problem:

1. Add a Text file named, answer.txt and in it, type your answer to the following questions:
    a. Is there Polymorphism in this project code. Yes or No.
    b. If you answered, Yes, then give the name of the Polymorphic method.

2. You must use the *StringQueue* class provided, whenever a queue is needed in your code.
3. The output of the Admin method *hourlyCompanyMessage* must be formatted as it has been done in the samples shown above.
4. Your *hourlyCompanyMessage* must call the *format*() method to perform the necessary formatting of messages.
5. The flow of your code, when *hourlyCompanyMessage* is executed, must match the sequence diagram shown below.
6. 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).
7. The *String* returned from *hourlyCompanyMessage*() in *Admin* must have the following format (which is produced by the *format*(*name*, *msg*) method):

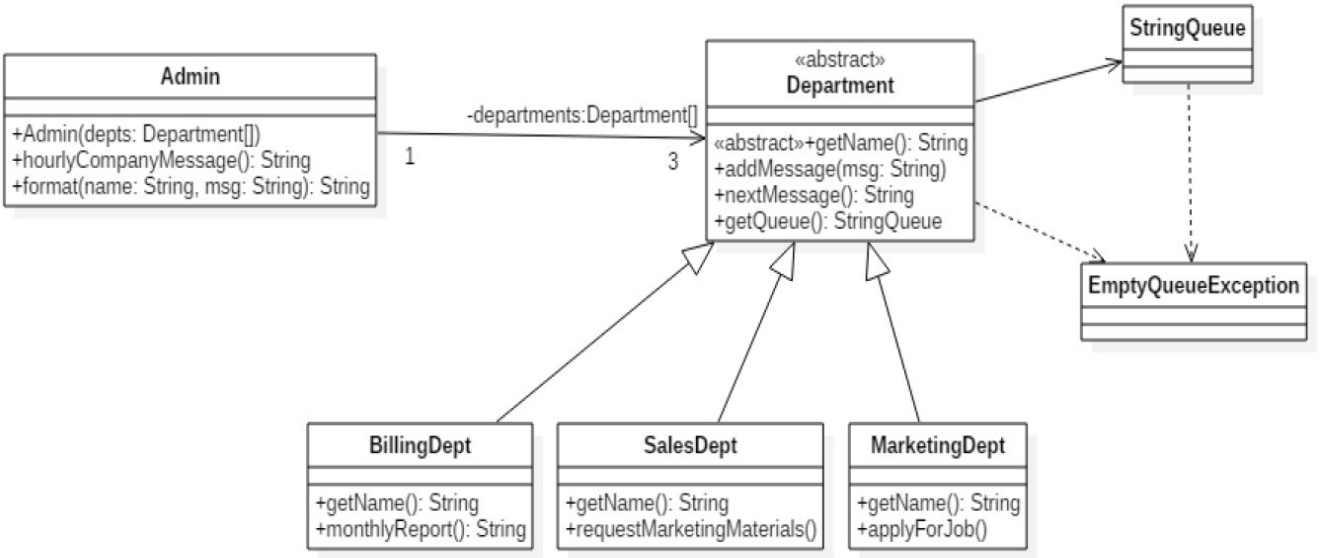
    <department_name>: <message>

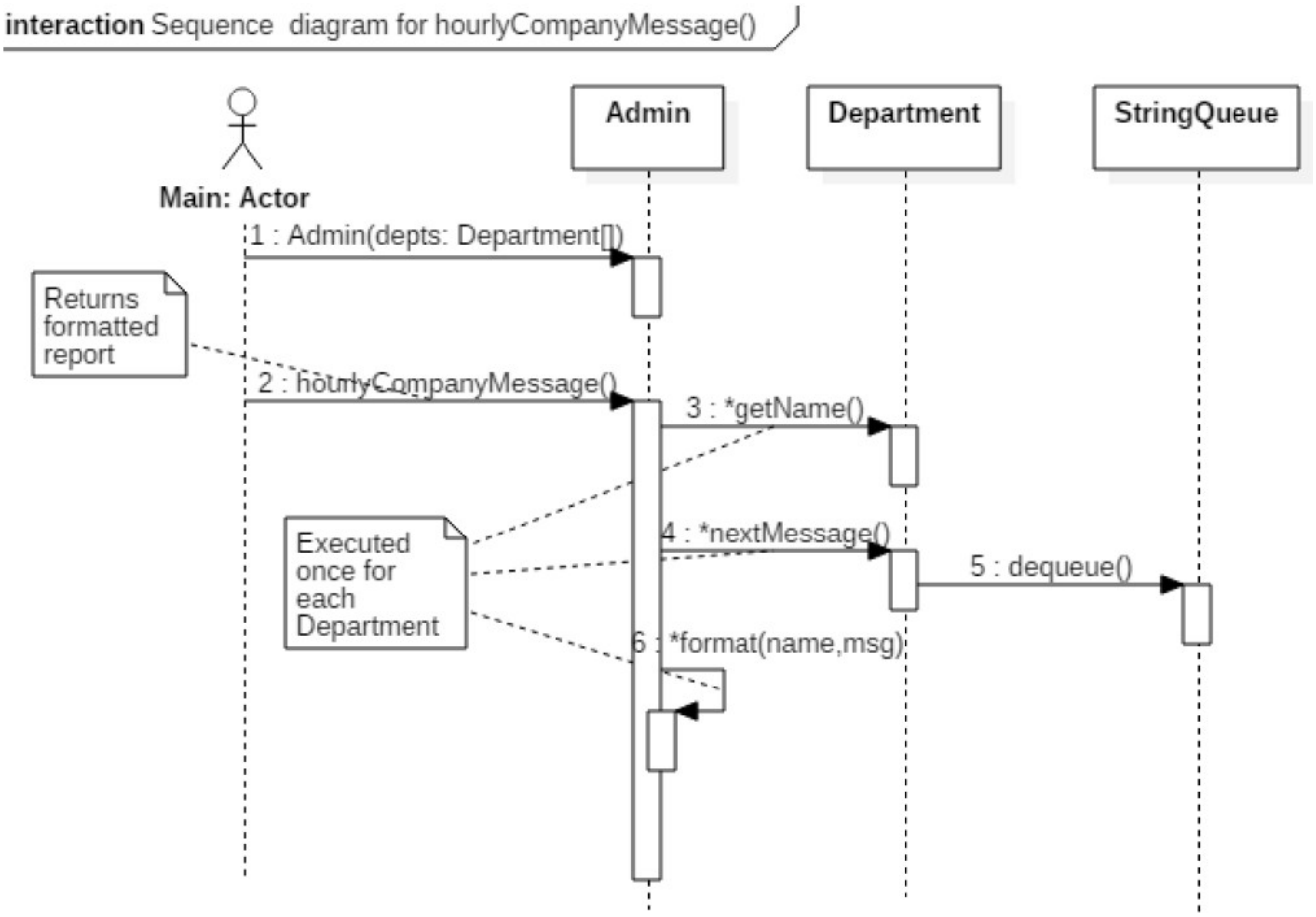
    For example:

    Billing: This is a message from the BillingDepartment

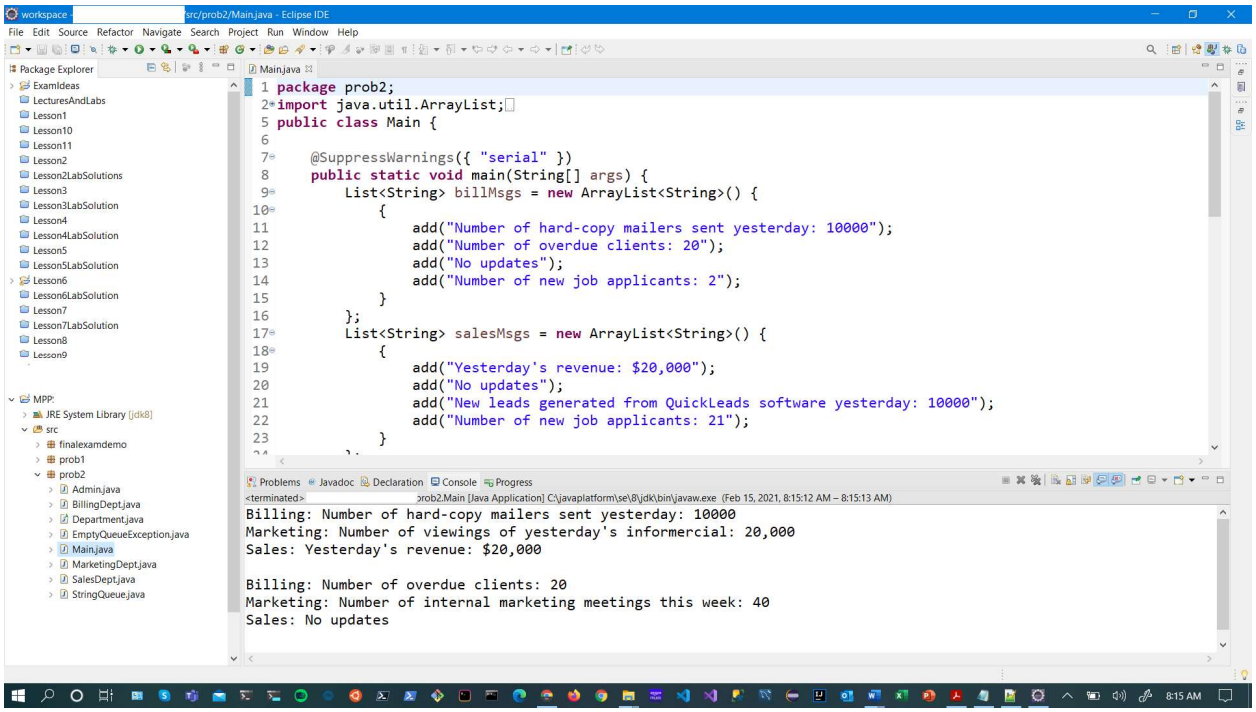
    The department name that appears in the output is obtained by calling the getName() method.

8. 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.) 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).
9. And finally: **Your submitted code must not have compiler errors or runtime exceptions when executed**.


Admin
+Admin(depts: Department[])
+hourlyCompanyMessage(): String
+format(name: String, msg: String): String
-departments:Department[]
1
3
«abstract»
Department
«abstract»+getName(): String
+addMessage(msg: String)
+nextMessage(): String
+getQueue(): StringQueue
StringQueue
EmptyQueueException
BillingDept
+getName(): String
+monthlyReport(): String
SalesDept
+getName(): String
+requestMarketingMaterials()
MarketingDept
+getName(): String
+applyForJob()


interaction Sequence diagram for hourlyCompanyMessage()
Main: Actor
Admin
Department
StringQueue
1 : Admin(depts: Department[])
Returns formatted report
2 : hourlyCompanyMessage()
3 : *getName()
Executed once for each Department
4 : *nextMessage()
5 : dequeue()
6 : *format(name,msg)

Shown below is a sample computer screenshot displaying the IDE window with code and result (in output console).



Sample computer screenshot displaying the IDE window with code and result (in output console):

//-- The End --//