**Project: Customer Support System**

**Summary:**

Develop a CONSOLE APPLICATION that can help companies guide their customer service tickets and make sense of their customer service conversations.

**Basic Functionalities:**

1. Handling customer services tickets
2. Storing large bodies of text in a structured way.
3. Providing basic analytics about the calls.
4. Helping to channel customers’ request.

**Users:**

1. Customer service representative
2. Customer services analyst

**Design**

1. Ticket:
   1. It is a super class, with three subclasses (PhoneCall, Email, InPerson).
   2. These subclasses all share basic ticket info (i.e. String ticketID, String customerRepID, String dateCreated, String customerServiceRepID), customer information (String id, String name, int phonenumber, String Address).
   3. Also, some basic information about the service that the customer has contacted the company about (String serviceID, String serviceStartDate, String serviceEndDate,),
   4. and some information about if the contact (boolean isResolved, and int departmentID). departmentID holds the ID of the department that the customer service transfers the ticket to. If the phone call is resolved, then the department ID will be 0. Otherwise, it is something between 1 and 10 (for different departments) or -1 (if the ticket is suspended).
   5. Finally, a Transcript object and a contactHandler method.
   6. Each subclass, however, has a different contactHandler method.
      1. For the emails, all the information is fetched to the system automatically, and the customer service rep only needs to view them and determine a next step for the ticket. Therefore for the Email class, the contactHandler method receives customer information, and service information, as input arguments. It also receives the actual text of the email as an argument. It then does two things: (1) prints the information and the text on the screen for the customer service representative and allow them to identify if the Ticket isResolved, or route it to a new departmentID. (2) creates a Transcript object from the text.
      2. For the in-person tickets, the customer service representative fills the customer and service information while the customer is in the store, but makes a decision about the ticket and writes down what he/she remembers from the conversation only after the customer leaves the store. The contactHandler method of the InPerson class does the same. The customer service representative needs to be able to update the customer and service information while the customer is present. When the customer leaves the store there is no need to update those info anymore, and only then, first, the representative should make a decision about isResolved and departmentID, and, secod, a Transcript object should be created as he/she types in what she remembers from the conversation.
      3. For phone conversations, the representative is constantly typing. He/she is typing what the customer says, and enters or updates their name or address or service info in between. The contactHandler should be able to do the same. A representative should always finish a sentence (by clicking enter) before being able to update information.
2. Transcript:
   1. The Transcript object stores the text in an ArrayList sentencesList, with each element being another ArrayList wordsList of type Strings. In other words, the text stored in both words and seentences. There is one ArrayList of sentences (ArrayList sentencesList) which stores the sentences. Each sentence is stored in the form of an ArrayList of Strings (ArrayList wordsList) with each element being a word of that sentence.
   2. Each word only consists of alphabets or numbers. No full stops, no semicolumns, no parentheses, etc.
   3. The transcript class has a public final statice array of type String which stores the stopWords in English, a public final static array of type String keeping a number of posetiveLexicons and a public final static array of type String keeping a number of negativeLexicons (see Appendix 1 for the list of each of these groups of words).
   4. All of the stop words in the sentences are removed.
   5. Each Transcript has a two dimensional frequencies[totalWordCount][2] array in which frequencies[totalWordCount][0]s are words and of type String and frequencies[totalWordCount][1]s are frequency of that word in that text and of type double.
   6. Each Transcript object has an averageSentenceLength attribute of type double, and two get and set methods for accessing and calculating that.
   7. Each Transcript object has a topFreqWords[3] of type string keeping the top 3 most used words, and two methods for getting and setting that.
   8. Each Transcript object has a getSimilarity method which gets a Transcript object in its argument and checks the similarity between most frequent words of that Transcript object (i.e. inputTranscript.topFreqWords) with its own object (i.e. this.topFreqWords[]).
3. FlowMonitor:
   1. FlowMonitor has a public static array of arrayRepresentativeIDs, arrayServiceIDs
   2. The FlowMonitor class manages the execution flow of the program.
   3. A flow manager class has an ArrayList of type Ticket.
   4. Every time that there is a new email, in-person, or phone contact with the company, the FlowMonitor, creates a new Ticket.
   5. FlowMonitor has an updateTickets method. After each ticket is created, the flow monitor loops over all of the tickets in the array to make sure there is no suspended ticket. If it finds any suspended ticket, it checks the similarity between its three more frequent words of that ticket with the three most frequent words of all other tickets in the array, and sets the departmentID of that ticket to be the same as the departmentID of the ticket with highest similarity.
   6. FlowMonitor has a getSentiment method which returns an integer between 1 and 3, 1 being angriest conversation and 3 being happiest conversation.
   7. FlowMonitor has a sentimentAnalysis method, which receives no argument and returns an ArrayList of three objects. Object one is an integer, which is the overall sentiment scores of the Tickets, object two is the average sentiment scores of the ticket for each customer representative, and object three is the average sentiment scores for the services.
   8. FlowMonitor has a printAnalysis methods that prints the results of one of the three analyses in 3.g. The method receives an integer between 0 to 2 (corresponsing to different analyses) and prints the requested analysis on the screen.
   9. FlowMonitor has a printFlowStatus method that prints all tickets in the system, their dateCreate, isSolved, and their status in a readable format.
4. Execution flow:
   1. The program waits for the representative to enter the type of contact (“phone”, “email”, “inperson”) or for the analyst to type “analysis”.
   2. If phone, email or inperson is selected, a Ticket of that type is created, and the program goes back to wait for a new contact.
   3. If analysis is selected, a menu with one item for each type of analysis, created in the FlowMonitor class in 3.g, another item for printing the flow status, and another item for updating the status of the tickets (calling updateTickets in 3.4)

**Note:**

1. The above design contains the main structure of the project. For the objects/data/methods that the design has not explicitly addressed them you are free to design/implement them as you think is the best.
2. I suggest that you first develop the UML together, identify the dependencies between the classes. Then, you can plan your project accordingly to minimum the time that one or two group members have to wait because their task depends on the tasks that should be done by another group member.
3. To minimize the wasted time, you can take one of the below alternative approaches:
   1. Start by developing classes with only shells for methods. Your group members can use those shell classes to develop their own parts, and gradually replace them with implemented classes.
   2. Use one of the co-development tools mentioned in the first session of the class (refer to the material for the first session on blackboard)
   3. Develop the parts of your project that are not dependent to your group members work, and remain the parts that are related unimplemented. Wait for your group members work to implement the dependent parts.
4. You need to test your code while writing it. Therefore, during the intial development phase, it is better if you have a main class in a Main.java file and the class that you are developing in a separate file. In the main method you can write simple instructions to test your methods. When you finalize the beta version of your implementation, you can just add the class to your final project which includes classes developed by your group members and finalize the project.
5. While it might not appear to many of you, you will need about one week for sitting together, putting the project together, and finalizing the development. Make sure that you start early and work on schedule so you can make the deadline.

**Appendix 1:**

Stop words:

a  
about  
above  
after  
again  
against  
all  
am  
an  
and  
any  
are  
aren't  
as  
at  
be  
because  
been  
before  
being  
below  
between  
both  
but  
by  
can't  
cannot  
could  
couldn't  
did  
didn't  
do  
does  
doesn't  
doing  
don't  
down  
during  
each  
few  
for  
from  
further  
had  
hadn't  
has  
hasn't  
have  
haven't  
having  
he  
he'd  
he'll  
he's  
her  
here  
here's  
hers  
herself  
him  
himself  
his  
how  
how's  
I  
I'd  
I'll  
I'm  
I've  
if  
in  
into  
is  
isn't  
it  
it's  
its  
itself  
let's  
me  
more  
most  
mustn't  
my  
myself  
no  
nor  
not  
of  
off  
on  
once  
only  
or  
other  
ought  
our  
ours

ourselves  
out  
over  
own  
same  
shan't  
she  
she'd  
she'll  
she's  
should  
shouldn't  
so  
some  
such  
than  
that  
that's  
the  
their  
theirs  
them  
themselves  
then  
there  
there's  
these  
they  
they'd  
they'll  
they're  
they've  
this  
those  
through  
to  
too  
under  
until  
up  
very  
was  
wasn't  
we  
we'd  
we'll  
we're  
we've  
were  
weren't  
what  
what's  
when  
when's  
where  
where's  
which  
while  
who  
who's  
whom  
why  
why's  
with  
won't  
would  
wouldn't  
you  
you'd  
you'll  
you're  
you've  
your  
yours  
yourself  
yourselves

Positive lexicons:

great

grateful

good

helpful

thankful

best

better

best

beyond

comfortable

commitment

committed

renew

register

unbelievable

unbelievably

awesome

wonderful

Negative lexicon

bad

worse

worst

crazy

confuse

confused

confusing

disturb

disturbance

disturbed

angry

annoy

annoyed

annoying

annoyingly

refuse

refused

regret

reject

rejected

repetitive

repetitively

cancel

discontinue

terminate

mad

unhappy

unhelpful

unlucky