Faculty of Engineering

Software Systems Engineering Program

2021 ENSE 475/ 375 Final Take-home exam

General Instructions:

1. Make sure you signed and returned the integrity form to the instructor before April 26 5:00.
2. The exam questions will be posted to UR courses at exam time.
3. Questions about the exam will be limited between 10:00 and 13:00 on exam day. To ask questions during that time, post your questions in the discussion forums so that All of you can check answers. There will be NO zoom sessions to ask questions.
4. You can make any reasonable assumptions but you have to justify them.

Coding and Testing

5. Write the code implementation yourself and don't use any library to do the work for you.
6. Follow TDD in your development of the code.
7. As part of the exam, you will download, configure and use a Jenkins plugin (a couple of MB)
8. Prepare a Jenkins CI/CD pipeline for the new application project. Use the pipeline to build the project, runs the tests, and deploy it to Docker hub. You have to submit information about your Jenkins environment/pipelines with suitable snapshots in your report.
9. FYI: we will run your code and test scripts through code similarity checkers.

Submissions

10. You submit a report showing your design of test cases. You can type it on the computer, or use a pen and paper and submit a scanned PDF file.
11. You have to submit (only with instructor) the GitHub (code and test scripts) and Docker Hub URLs you used for your implementation (keep them private until the due time is over).
12. A required submission for this exam is to video record your screen for every step you make and command you issue. So, prepare a screen recording software for your computer. Submit a video link to your video.
13. We will mark over two rounds. The first round to gauge the average (and difficulty of this exam) for the whole class. The second round we mark based on our estimate of the average.
14. The marking rubric will look something like the following table

Step	Max Mark	Class 1	Class2	….	Class 6	Marking remarks	Final mark
Design Tests	40%	Student mark					
Write test scripts	15%						
Write Code	30%						
Evaluate Tests	10%						
Time zone Calculation	10%						
Total	105%						

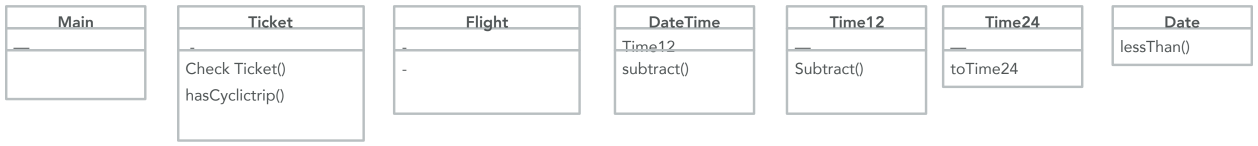
University of Regina Undergraduate Calendar (p. 36) Cheating constitutes academic misconduct. Cheating is dishonest behaviour or the attempt to behave dishonestly. It includes, but is not limited to:

- Using books, notes, diagrams, electronic devices, smart devices, or any other aids during an examination, either in the examination room itself or when permitted to leave temporarily unless explicitly authorized by the course instructor or examiner.
- Copying the work of other students.
- Communicating with others during an examination to give or receive information, either in the examination room or outside it or through the use of electronic communication devices.
- Consulting others on a take-home examination unless authorized by the course instructor.
- Commissioning or allowing another person to write an examination on one's behalf.
- Not following the rules of an examination.
- Using for personal advantage, or communicating to other students, advance knowledge of the content of an examination.
- Altering answers on an assignment or examination that has been returned.
- Removing an exam or exam related materials from the examination room if not permitted to do so.

Exam Questions.

For this exam, you write code and test a part of an airline ticketing application. <u>**You follow TDD**</u> to write and test the code of these methods. You evaluate the tests after each cycle of TDD.

The application consists of a number of classes. Parts of code for these classes is provided within a compressed file at UR courses. The code is commented. Some code for some methods is missing and their locations are highlighted. To complete the code for these methods, you read the following specifications of the application. You design test cases for the different methods. The following partial class diagram lists application classes and the methods to be completed.



<u>**Application Specifications:**</u> The application (*class: APP – method: main*) checks if an airline ticket meets the requirements of the user. A ticket consists of a list of flights. Each flight has departure airport, departure time, arrival airport, arrival time. The main method collects the details of flights from std input. In addition, the main method collects users' preferences about a ticket as follows:

- maximum number of flights preferred
- maximum total flight time preferred (Total flight time is calculated as the summation of the duration of each flight.)
- maximum total layover time preferred (Total layover time is the summation of all layover times. [A layover time is the waiting time between arrival time into an airport and departure time from that airport]

The application represents an airline ticket in (*class: Ticket*) and consider it to be as a list of ordered flights (*class: Flight*). Flight information include: Departure airport, Departure Date & Time, Arrival airport, Arrival Date & Time. IATA 3-characters standard code (eg YQR, for Regina airport) is used to represent airports. A (class: DateTime) is used to represent the date & time of arrival or departure of a flight. The class combines two objects of types (class: Date) to represent the date and (class: Time12) to represent time. Time12 objects represent time in a H:M AM/PM format, where H is between 1 to 12 and M is between 0 and 59 followed by -AM/PM; Seconds are not used. Time12 format is considered the standard format, for the purposes of this exam, for expressing times for UI.

1st Iteration development: The application checks many constraints about tickets using (class: Ticket-*method: CheckTicket)* including:

- Airport codes are formatted according to IATA standard (that is, 3-character codes).
- Check if the arrival airport of a flight is the same as the departure airport for the next flight.
- Checks for Schengen area (https://en.wikipedia.org/wiki/Schengen_Area) airports as there can be no flight from a Schengen airport to another Schengen airport without a Schengen visa (A list of Schengen airports is given in the code).

Another very important constraint to check if the ticket meets user preferences. This is carried over many classes for modularity of code.

1. the method (*class: Flight - method: calculateFlightTime*) calculates the flying time of each flight. The summation of this has to be less than or equal to max flight time identified as a user preference.
2. the method (*class:Ticket - method: calculateLayoverTime*) calculates the layover times between each two flights. The summation of all layover times should be less than the max preferred by the user.

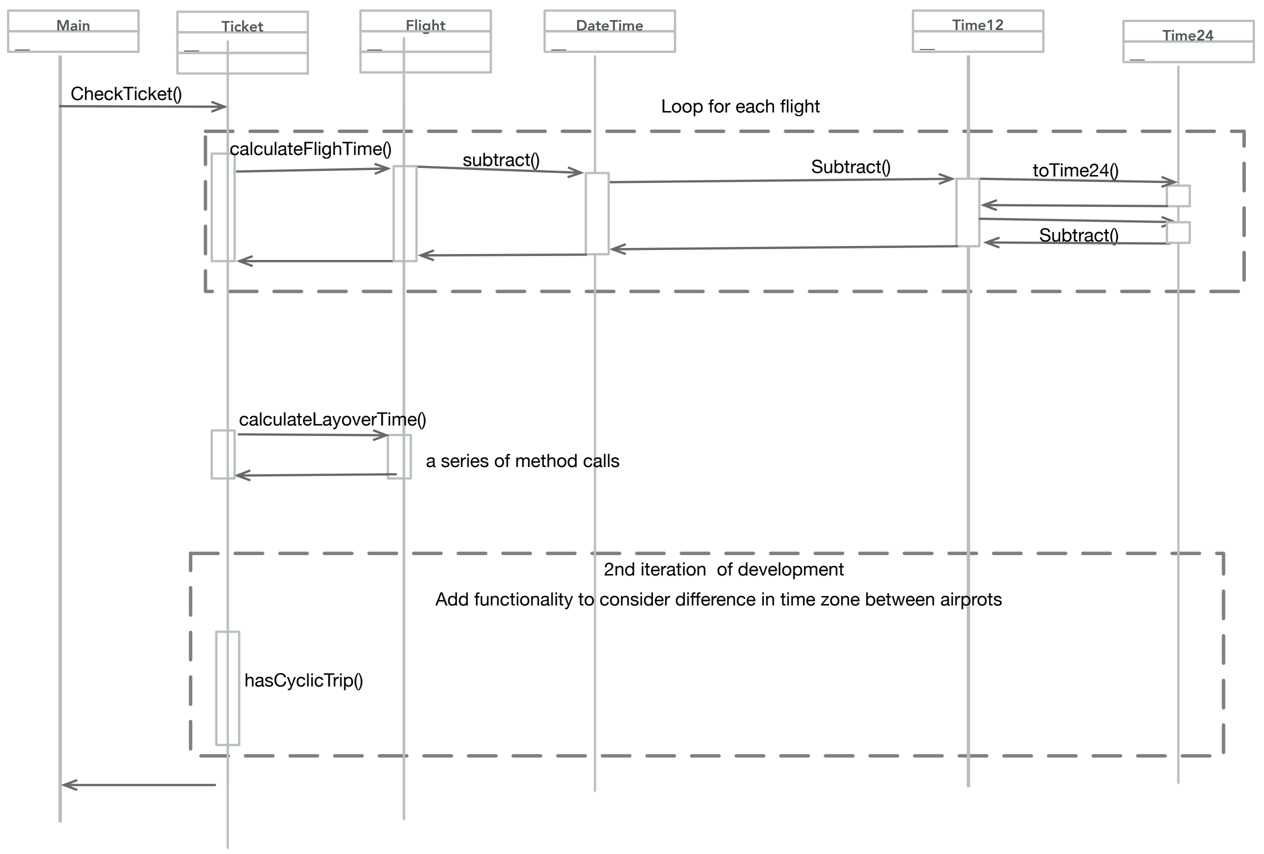
The method *(class: DateTime – method: subtract)* is used to calculate the time difference (in minutes) between two DateTime objects. The method *(class: DateTime – method: subtract)* uses (class: Date) and (class:Time12) functions in its time calculations. For simplicity:

- let's assume, the method works only with no more than two consequent days and throw an exception if the two dates are not in the same day or in two consecutive day.
- Let's do time difference calculations in 24 hours format represented by (class: Time24). (Time12 formats are a bit more complex). Time24 objects express time in a H:M where H is between 0 and 23. The method *(class:Time24 - method: toTime24)* does this conversion. Time24 is used by developers to make internal calculations easier and not used in communicating with user.

2nd Iteration development: _Once you are done with the above, you can:

- Add code to consider different time zones between airports.
- Add code that does not allow cyclic trips in a ticket (that is, no passenger can arrive at the same airport more than once within the same ticket). A round trip is allowed though (very first departure airport can be the very last arrival airport).

The following illustrative partial (not complete) sequence diagram represents how flight time is checked and calculated and how layover time is calculated.



Questions:

1- Add to Jenkins a "Test Results Analyzer" plugin to help you in visualizing the tests results.
2- For the method: *(class:Time24 - method: toTime24)*
   - Implement the function.
   - draw a CFG for your implementation, and then identify paths required for 100% node coverage, 100% edge coverage, 100% edge-pair coverage and identify all prime paths.
   - Build a test case table where each path you identified above corresponds to a testing case.
   - Write a Junit test script for this function
   - Update Jenkins pipeline to check this Junit test in a separate node.
3- For the method: (class:Time12 – method: subtract)
   - Carry out an <u>input domain modelling</u> and clearly state the characteristics, domains, blocks, values. Use the pairwise coverage criteria.
   - Implement the function by converting the 12-houe time first to 24-hour time.
   - Write a Junit test script for this function
   - Update Jenkins pipeline to check this Junit test in a separate node.
4- For the method: (class:Date – method: LessThan)
   - Carry out an <u>input domain modelling</u> and clearly state the characteristics, domains, blocks, values. Use the pairwise coverage criteria.

- Implement the function by converting the 12-houe time first to 24-hour time.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

5- For the method: (class:DateTime – method: subtract)
- Implement the function.
- draw a CFG for your implementation, and then identify paths required for 100% node coverage, 100% edge coverage, 100% edge-pair coverage and identify all prime paths.
- Build a test case table where each path you identified above corresponds to a testing case.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

6- For the method: (class: Ticket - method: CheckTicket)
- Implement the checkTicket function in the Ticket class.
- draw a CFG for your implementation, and then identify paths required for 100% node coverage, 100% edge coverage, 100% edge-pair coverage and identify all prime paths.
- Build a test case table where each path you identified above corresponds to a testing case.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

2nd Iteration development

7- For the method: (class:Ticket - method: hasCyclicTrip)
- Carry out an input domain modelling and clearly state the characteristics, domains, blocks, values. Use the pairwise coverage criteria.
- Implement the function.
- Write a Junit test script for this function
- Update Jenkins pipeline to check this Junit test in a separate node.

8- Implement functionality to consider different time zones at different airports.

==≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠≠End.