Assignment 2

Weight: 30% of your final grade

Due: after Unit 7

This assignment comprises 10 questions. Questions 1–7 are worth 20 marks each, and questions 8–10 are worth 40 marks each. Answer any **three** questions from questions 1–7 and only **one** question from questions 8–10. You are expected to complete this assignment before you start Unit 8.

Upload your completed assignment to the Assignment 2 link on the course home page for marking and tutor feedback.

☞ Be sure to complete the final step—click on the **Send for Marking** button to notify your tutor.

Prerequisite: Read what an API is in the textbook *Introduction to Programming Using Java* by David J. Eck on pages 142–143.

When solving the problems in this assignment, you must follow the application programming interface (API) expected in each problem. You should implement all the **attributes** and **operations** mentioned in the API.

Notice that there is no `main` method in the APIs. That is, you should not perform any data processing within the `main` method. You should rather use the `main` method to test other methods, prompt the user for some inputs, and display the results returned by your methods.

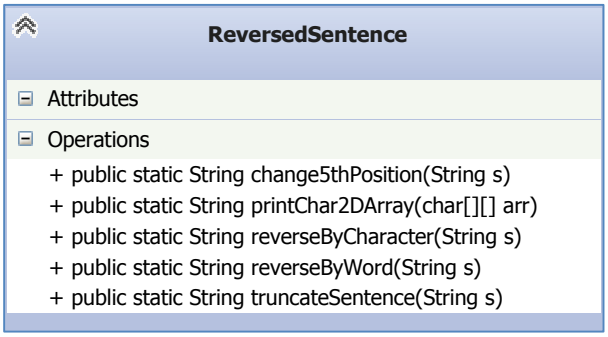
1. [20 marks] Read three sentences from the console application. Each sentence should not exceed 80 characters. Then, copy each character in each input sentence in a [3 x 80] character array.

   The first sentence should be loaded into the first row in the reverse order of *characters* – for example, “mary had a little lamb” should be loaded into the array as “bmal elttil a dah yram”.

   The second sentence should be loaded into the second row in the reverse order of *words* – for example, “mary had a little lamb” should be loaded into the array as “lamb little a had mary”.

   The third sentence should be loaded into the third row where if the index of the array is divisible by 5, then the corresponding character is replaced by the letter ‘z’ – for example, “mary had a little lamb” should be loaded into the array as “mary zad azlittze lazb” – that is, characters in index positions 5, 10, 15, and 20 were replaced by ‘z’.

   Note that an empty space is also a character, and that the index starts from position 0. Now print the contents of the character array on the console.



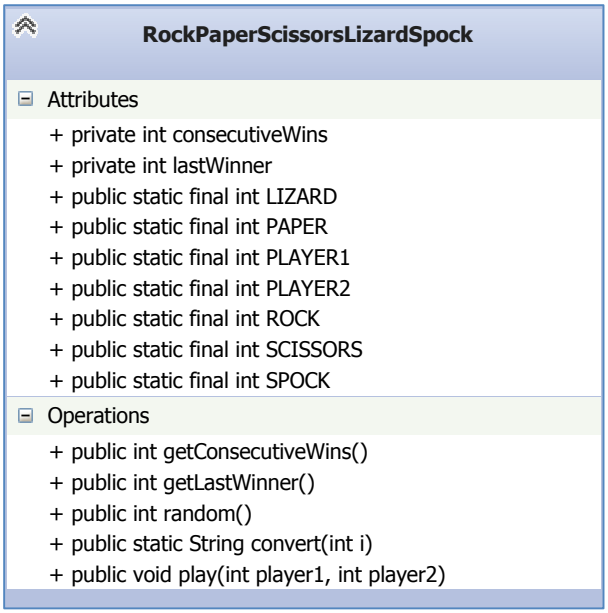
2. [20 marks] Write a program that plays the Rock-Paper-Scissors-Lizard-Spock game. Refer to http://en.wikipedia.org/wiki/Rock-paper-scissors-lizard-Spock for more information.

   Normally, one player is a human and the other is the computer program. However, in this exercise, the program will generate two players who play against each other. The play continues until either of the computer-generated players wins *four* consecutive times.

   In this game, two random integers are generated in the range of [1 to 5], one per player. 1 refers to Rock, 2 refers to Paper, 3 refers to Scissors, 4 refers to Lizard, and 5 refers to Spock.

   For example, if the computer randomly generates integers 2 and 5 in the first iteration, 2 is for the first player and 5 is for the second player. Based on Rule 8 in the following 10 rules, Paper (2) disproves Spock (5), so Player 1 wins. Repeat it to generate one more pair and determine who wins that iteration. Continue the iterations until one player wins four consecutive times.

   Rule 1: Scissors cut paper
   Rule 2: Paper covers rock
   Rule 3: Rock crushes lizard
   Rule 4: Lizard poisons Spock
   Rule 5: Spock smashes (or melts) scissors
   Rule 6: Scissors decapitate lizard
   Rule 7: Lizard eats paper
   Rule 8: Paper disproves Spock
   Rule 9: Spock vaporizes rock
   Rule 10: Rock breaks scissors



3. [20 marks] Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with 4 for Visa cards, 5 for Master cards, 37 for American Express cards, and 6 for Discover cards. In 1954, Hans Luhn of IBM proposed the following algorithm for validating credit card numbers:
   a. Double every second digit from right to left (e.g., if number is 3 => 3 * 2 => 6) and add them together.
   b. If this doubling results in a two-digit number, then add the two digits to get a single-digit number (e.g., if number is 5 => 5 * 2 => 10 => 1+0 => 1).

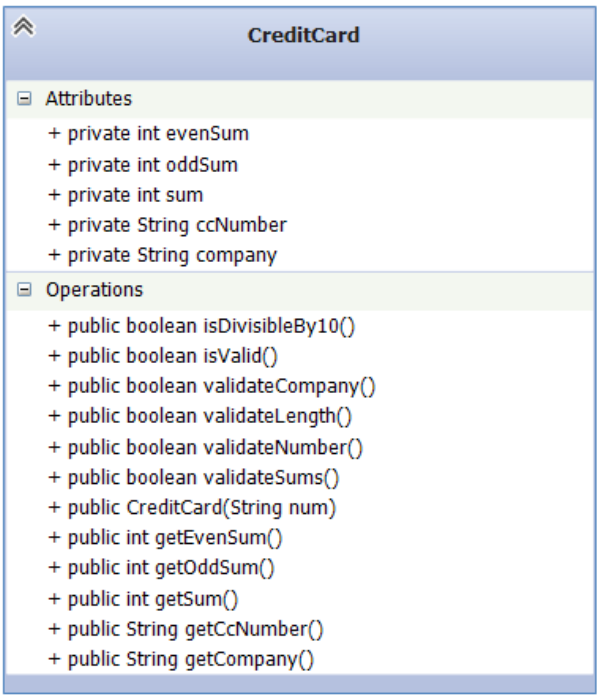
   So, for the credit card number 4388576018402626, doubling all second digits from the right results in (2 * 2 = 4) + (2 * 2 = 4) + (4 * 2 = 8) + (1 * 2 = 2) + (6 * 2 = 12 = 1 + 2 = 3) + (5 * 2 = 10 = 1 + 0 = 1) + (8 * 2 = 16 = 1 + 6 = 7) + (4 * 2 = 8).

   This totals to 4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37. Add all digits in the odd places from right to left.

   The leftmost digit of the credit card number is at index 0; 6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38.

   Add results from steps (a) and (b) and see if divisible by 10. If it is, then the card number is valid; otherwise invalid. 37 + 38 = 75 is not divisible by 10, so it is an invalid credit card number.

   Implement Luhn's algorithm in a program to determine whether a given credit card number is valid or not. You must test if the number of digits in the input is in the valid range (13 to 16), run Luhn's algorithm to test its validity, and if it is valid, print the name of the company that offers that credit card number.



4. [20 marks] Craps is a dice game where two dice are rolled. Each die has six faces representing values 1, 2, 3, 4, 5, or 6.

   I. If the sum is 2, 3, or 12 (called *craps*), you lose;

   II. If the sum is 7 or 11 (called *natural*), you win;

   III. If the sum is any other value (4, 5, 6, 8, 9, or 10), a value point is established, and you continue to roll until you either roll a sum of the value point or a 7. If the sum of the new roll is equal to the value point, then you win; if the sum of the new roll is equal to 7, then you lose.

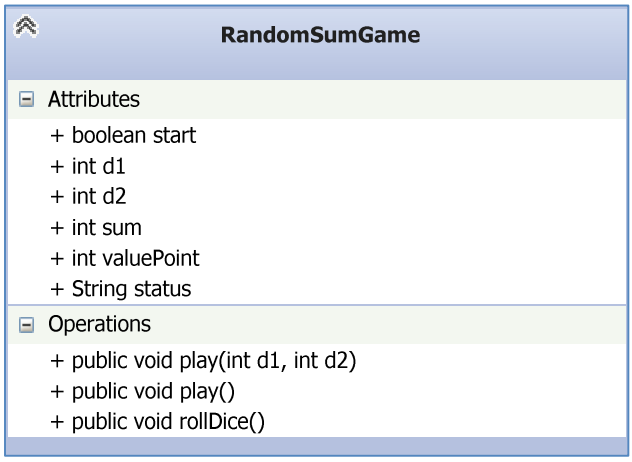
   Remember, in option (III), you continue to roll until you get a 7 or the value point.

   Sample runs:

   - You rolled 5 + 6 = 11; you win
   - You rolled 1 + 2 = 3; you lose
   - You rolled 2 + 2 = 4; you establish the value point 4;
     - Roll again 2 + 3 = 5; roll
     - Roll again 2 + 1 = 3; roll
     - Roll again 2 + 2 = 4; you win
   - You rolled 2 + 6 = 8; you establish the value point 8;
     - Roll again 4 + 4 = 8; you win
   - You rolled 3 + 2 = 5; you establish the value point 5;
     - Roll again 1 + 1 = 2; roll
     - Roll again 2 + 2 = 4; roll

- Roll again 1 + 1 = 2; roll
- Roll again 3 + 4 = 7; you lose

Develop a program that plays craps with a player **three** times. At the end, the program prints the number of times the player won and the number of times the player lost.



5. [20 marks] Create three classes: `Village, Citizen`, and `ComputeIntellect`.

   The `Village` class has an instance variable called `numberOfCitizens` and an array that holds a maximum of 100 `Citizen` objects.

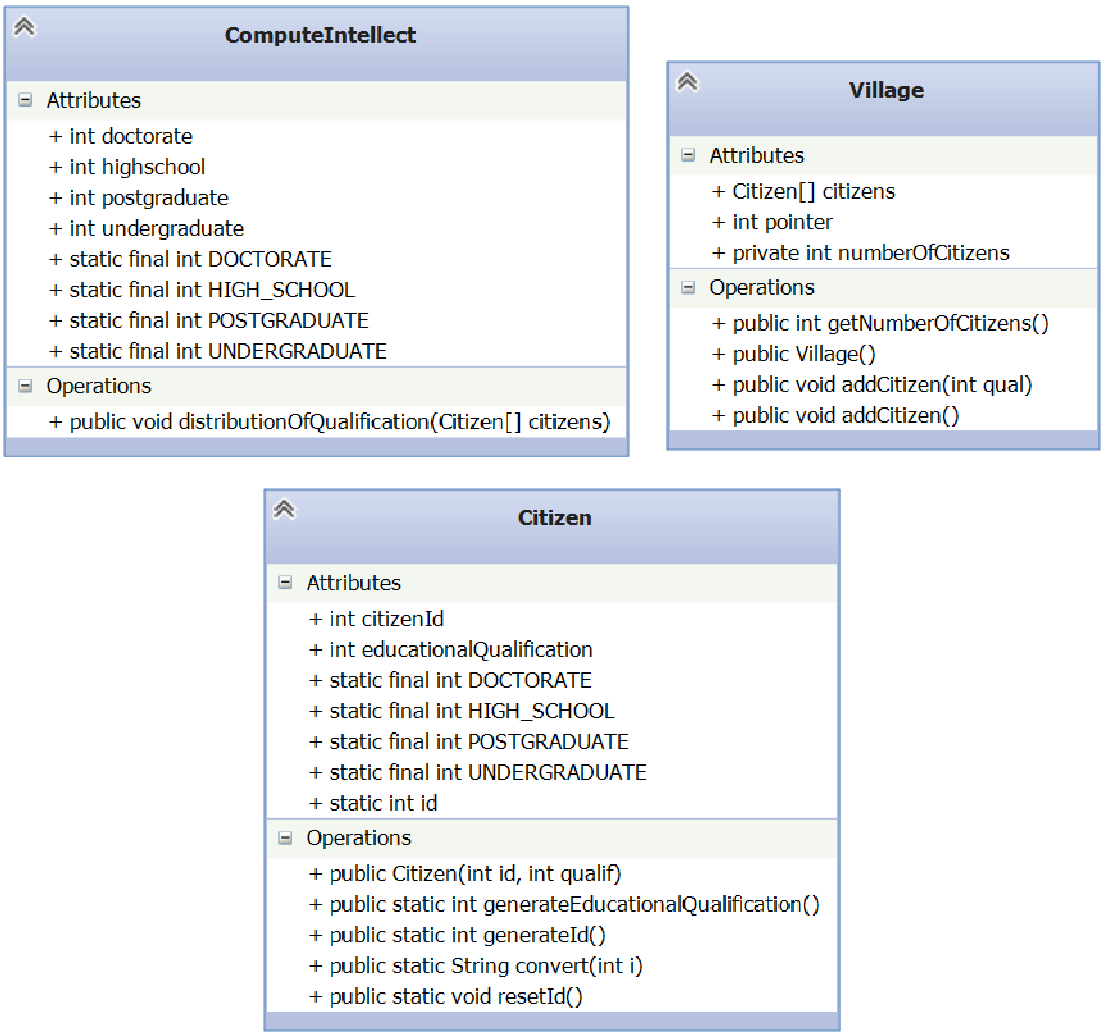
   The `Citizen` class has `citizenId` and `educationalQualification` as instance variables.

   The `ComputeIntellect` class has a `distributionOfQualification()` method.

   Create 100 `Citizen` objects using `citizenId` for the range [1 to 100]. Randomly generate the educational qualification in the range [1 to 4], where 1 = high school, 2 = undergraduate, 3 = postgraduate, and 4 = doctorate.

   Store these 100 objects in a `Village` object using an array – any array of your choice.

   The `distributionOfQualification()` method loops through the 100 objects and counts the number of citizens corresponding to each of the four educational qualifications.



6. [20 marks] Implement a Java method that prints out the day of the week for a given day (1. . . 31), month (1 . . . 12) and year in the range of March 1900 to February 2100.

   Calculate the day of the week for the dates between March 1900 and February 2100 as follows:

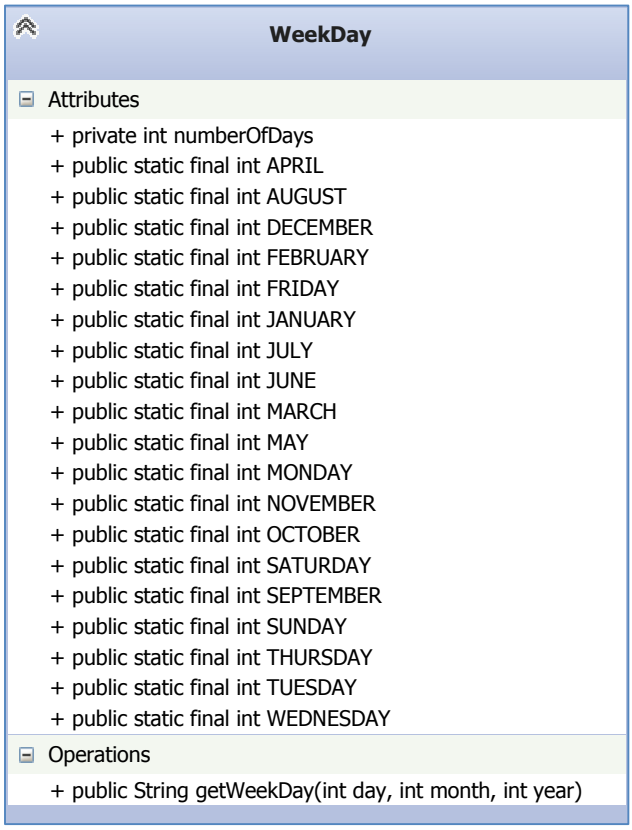
   First, you have to calculate the total number of days from 1900/1/1 to the given date (see below for details).

   Secondly, you divide this number by 7 with an integer remainder: This now is the day of the week, with 0 as Sunday, 1 as Monday, and so on.

   To calculate the total number of days, you have to implement the following steps:

   - Subtract 1900 from the given year, and multiply the result by 365
   - Add the missing leap years by adding (year − 1900) / 4.

- If the year itself is a leap year and the month is January or February, you have to subtract 1 from the previous result.
- Now add all the days of the months of the given year to the result (in the case of February, it is always 28 because the additional day for a leap year has already been added in the calculation).



7. [20 marks] Create a `Person` class that includes the name of the person, the weight of the person (in pounds), and the height of the person (in inches). For the data listed in the table below, create four `Person` objects. Compute their individual body mass index (BMI) and store it as part of these objects. Further, determine their weight category (see below) and add that information as part of the object as well. Store each of these four `Person` objects, their corresponding BMI, and weight category in a different `ArrayList` and develop `get` and `set` methods to access elements in that `ArrayList`.

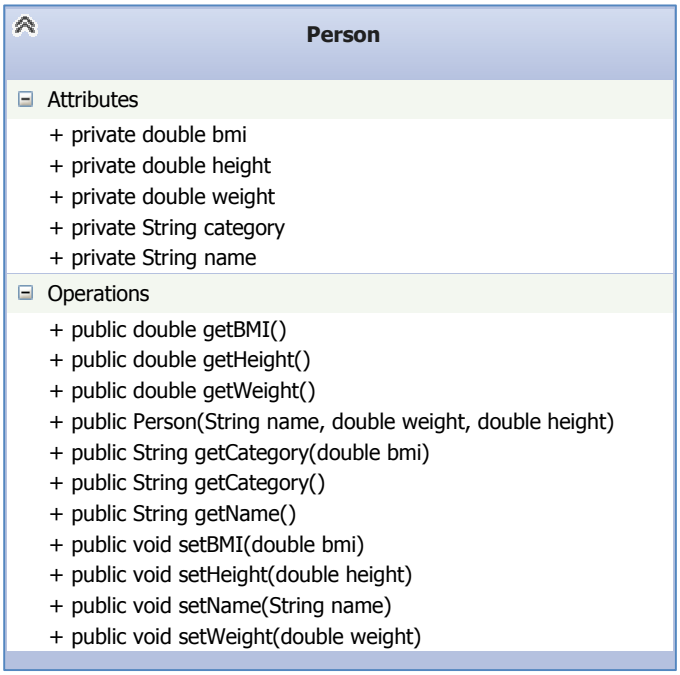
Name	Weight (pounds)	Height (inches)
Andrew	125.5	55.1
Boyd	150.0	67
Cathy	135	72.3
Donna	190	64

BMI is calculated using the following formula:

$$BMI = \frac{weight\ (lb) \times 703}{(height\ (in))^2}$$

BMI can indicate the following categories:

- Underweight when BMI is less than 18.5
- Normal weight when BMI is between 18.5 and 25
- Overweight when BMI is between 25 and 30
- Obese when BMI is 30 or greater



8. [40 marks] The following is a score of a single game in badminton. The top row is the score for Player 1. The second row is the score for Player 2. Represent this data in an `ArrayList` in a class called `BadmintonScoring`.

0	1	2						3	4										5						
0			1	2	3	4	5			6	7	8	9	10	11	12	13	14	15		16	17	18	19	20	21

I. Compute the maximum points scored by Player 1 and Player 2.

II. Compute the maximum number of points scored in a continuous sequence by Player 1 and Player 2. *Hint*: Player 1 scored the sequence 0-1-2, which implies s/he scored 2 points in a

continuous sequence. Similarly, for Player 2, 16-17-18-19-20-21 implies that s/he scored 5 points in a continuous sequence.

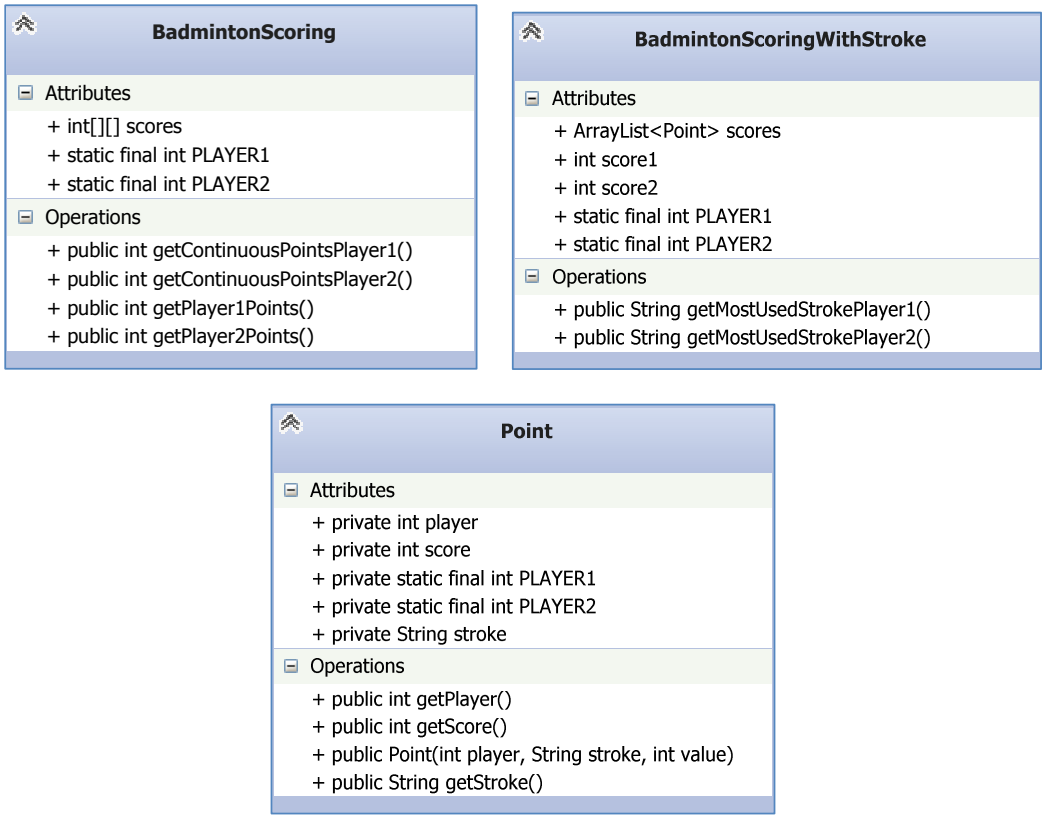
III. Extend `BadmintonScoring` to associate each point scored by a player with a particular stroke that earned that point, using the notion of *association list*. You can represent each point as an object and store the score of a player in an association list (refer to Chapter 7, section 7.4.2 for details). For example, when Player 1 scored his/her first point, instead of just 1, it could have been {1, slice}. Thus, each point is augmented with the type of stroke from the following list:

a. slice
b. drive
c. smash
d. drop
e. net-shot

IV. Store the following score of a single game using the modified `BadmintonScoring` class.

0	1 a	2 c						3 a	4 c											5 c						
0			1 d	2 e	3 d	4 e	5 d			6 e	7 e	8 a	9 d	10 e	11 e	12 e	13 e	14 e	15 e		16 e	17 e	18 e	19 e	20 e	21 a

V. Identify the type of stroke that earned most points for each player.



9. [40 marks] Create a 10x10 matrix as a 2D array. See a sample array below.

	0	1	2	3	4	…	9
0	R1[0, 0]	[0, 1]	[0, 2]	[0, 3]	[0, 4]	…	[0, 9]
1	[1, 0]	[1, 1]	[1, 2]	[1, 3]	[1, 4]	…	[1, 9]
2	[2, 0]	[2, 1]	[2, 2]	[2, 3]	[2, 4]	…	[2, 9]
3	[3, 0]	[3, 1]	[3, 2]	[3, 3]	[3, 4]	…	[3, 9]
4	[4, 0]	[4, 1]	[4, 2]	[4, 3]	[4, 4]	…	[4, 9]
…	…	…	…	…	…	…	…
9	[9, 0]	[9, 1]	[9, 2]	[9, 3]	[9, 4]	…	[9, 9]

Assume that a robot is placed in position [0, 0]. Now randomly generate a move. The move could take the robot to one of the eight possible adjacent slots – {up, down, left, right, left-up-corner, left-down-corner, right-up-corner, and right-down-corner} – these slots are represented by {1, 2, 3, 4, 5, 6, 7, 8}. However, at [0, 0], the robot only has three possible slots to move to – right, down, right-down-corner.

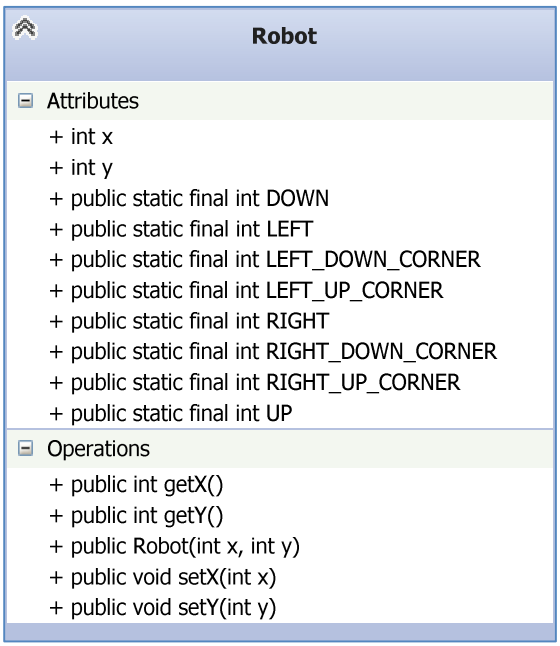
Create another robot called R2 and place it on [9, 9].

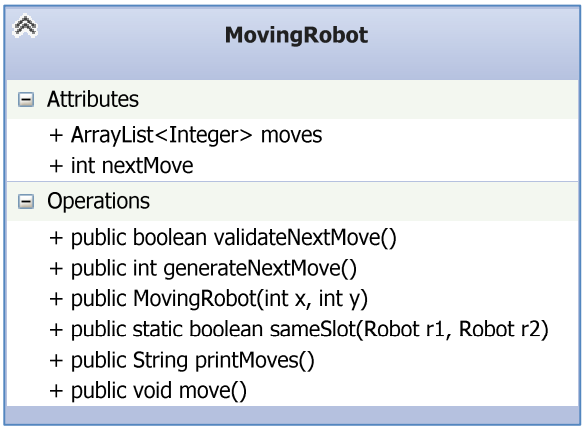
Now randomly generate an integer in the range of [1 to 8]. This first random integer corresponds to a possible move for Robot R1. If the move is valid, then move R1 to its new slot. A move is invalid if it takes the robot out of bounds of the [10x10] matrix. If the move is invalid, then keep generating random integers until a valid move is found.

Repeat this procedure for the second Robot R2.

If both R1 and R2 are in the same slot, then stop, print the final slot, print the sequence of random numbers that led R1 to this slot, and the print the sequence of random numbers that led R2 to the same slot.

Implement this program with a `Robot` class and a `MovingRobot` subclass.





10. [40 marks] A train timetable for a train travelling between Vancouver and Toronto is given below.

Station	Arrival	Departure	Day
Vancouver		20:30	1
Kamloops	06:00	06:35	2
Jasper	16:00	17:30	2
Edmonton	23:00	23:59	2
Saskatoon	08:00	08:25	3
Winnipeg	20:45	22:30	3
Sioux Lookout	05:02	05:42	4
Hornepayne	15:35	16:10	4
Capreol	00:18	00:48	5
Toronto	09:30		5

Store the information from each row of the table in an object. Then, arrange the objects in an `ArrayList` structure.

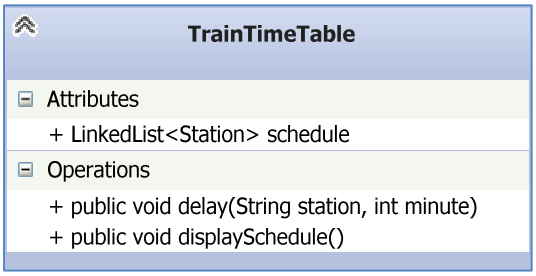
Your program should now take the following commands in a continuous loop:

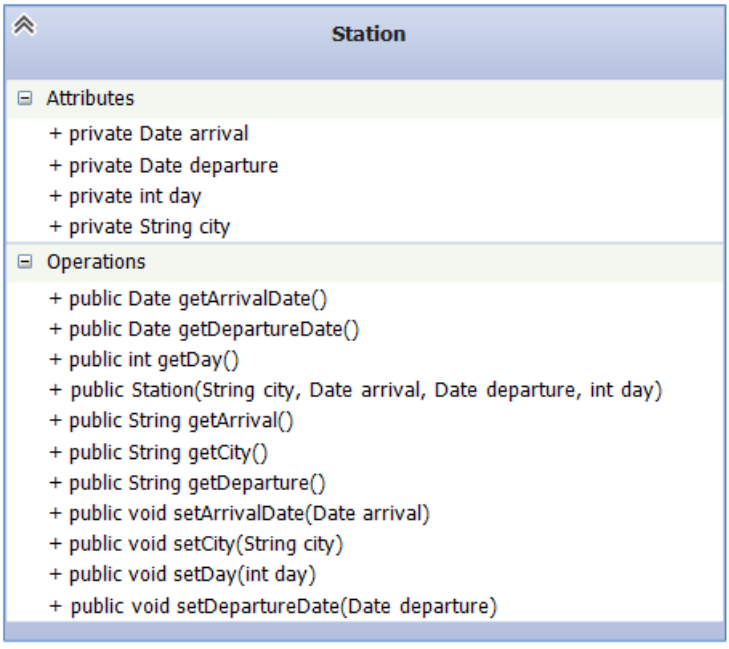
I. `Show` – shows the full table

II. `Delay` <station> <minutes> – the arrival of the train is delayed by <minutes> at station <station>; that is, add the delay to the corresponding station entry. For example, Delay Edmonton 30 implies that the train would arrive 30 minutes later than the expected time of arrival in Edmonton. The new entry would be "Edmonton 23:30 00:29 3". All stations following Edmonton will also update their arrival and departure by +30 minutes, and consequently the day of arrival and departure as well. The result of this `Delay` command is shown below:

Station	Arrival	Departure	Day
Vancouver		20:30	1
Kamloops	06:00	06:35	2
Jasper	16:00	17:30	2
Edmonton	23:30	00:29	3
Saskatoon	08:30	08:55	3
Winnipeg	21:15	23:00	3
Sioux Lookout	05:32	06:12	4
Hornepayne	16:05	16:40	4
Capreol	00:48	01:18	5
Toronto	10:00		5

III. `Quit` – stop the program from accepting any more commands.





Please note that class “Date” and class “Calendar” are mutable in Java. You are welcome to use either “Date” or “Calendar”, whichever seems easier for you to complete/solve the problem.