# COMP 1501 ASSIGNMENT 2

**Given:** Saturday, October 10, 2015

**Due Dates:** To be completed individually.

**Part 1: IPO diagram, "contract" (using MS Word) for each function, due data on or before Friday, October 16, 2015 11:59pm**

**Part 2: Implementation (in Java Code) due date on or before Sunday, October 25, 2015, 11:59pm**

## Points for early submission:

**Part 1:** No points will be awarded for early hand-in for Part 1.

**Part 2:**

- 0 points = October 25rd after midnight. Assignments handed in after midnight will be awarded zero points but will be accepted up **until 6am the following morning without any penalties**.
- 1 point = October 24rd before midnight.
- 2 points = October 23nd before midnight.
- 3 points = October 22st before midnight.
- 4 points = October 21th before midnight.

## Objectives:

- To continue to gain experience developing a complete Java program, including:
  - understanding the problem;
  - designing an algorithmic solution using methods;
  - coding and testing that solution.
- To continue to gain experience with arithmetic expressions, including the use of integer "div" and "mod" operators.
- To learn to develop programs that create objects and invoke their methods.
- To gain experience manipulating strings.
- To learn more about classes in Java's standard class library (`Scanner, Random, String,` etc).
- To gain experience searching and reading the Java API documentation.

## Problem Overview:

The automobile has just been invented in the fictional Kingdom of Westeros. When a car is purchased, it comes with a license plate in "LLL-DDD" format. For example, "ZHM-597". The law requires that the new owner register this vehicle with the Kingdom. The King has ordered you to write a program that generates vehicle registration information for new car owners. Each registration number has exactly 12 digits. For example:

```
510739940263
```

In addition, each registration has a renewal month in "`MMM yyyy`" format. For example, "`Oct 2013`" means renewal is required by the end of October 2013.

New registration numbers are generated randomly, subject to the following constraints:

1) The leftmost digit must be greater than 2. Digits 0 through 2 are reserved for special purposes by the Kingdom, so registrations numbers that lead with 0, 1 or 2 are never issued.

For example, above, the leftmost digit is 5 and so satisfies this constraint.

2) The rightmost digit is a "checksum" digit: it must equal the sum of digits 3, 6 and 8 (counting from the left), mod 10. This is used as a simple test for validity by police and other officials whenever registration is presented.

   For example, "510739940263" above gives 0 + 9 + 4 = 3 (mod 10), resulting in "510739940263".

3) The sum of the least significant two renewal year digits, mod 10, are encoded as digit 8. This is used as another simple test for validity.

   For example, an expiry year of 2013 is encoded as "510739940263".

4) The third digit is based on the middle letter of the license plate and the number portion of the license plate. The alphabetic position of the letter (1 for 'A', 2 for 'B', etc.) is added to the number to produce a sum. The middle digit of this sum is taken as the result digit. This is used as a third test for validity.

   For example, with a license plate number of "ZHM-597", 8 (for 'H') is added to 597 to produce 605. Thus, the tens digit of the sum is encoded in the registration number as "510739940263".

### Functional Requirements of Finished Program:

Your program must behave exactly as illustrated in the sample run below:

Enter license plate number: ***zhm-597***
Enter owner name: ***john snow***

Information for printed registration form:

| | |
|---|---|
| Registration No. | 510739940263 |
| License Plate No. | ZHM-597 |
| Owner | Snow, John |
| Renewal Month | Oct 2013 |

The program reads:

- a license plate number (in "LLL-DDD" format, i.e. three letters and three digits separated by a dash);
- the owner's first and last name.

The program writes back the information to be printed on the new registration form:

- a new random registration number (subject to the validity checks described above);
- license plate number (as above, but always capitalized)
- owner name (surname first, always with capital initials and remaining letters lowercase);
- an expiry date in "MMM yyyy" format.

The expiry date must be based on the current date (i.e. whenever the program is run) plus one year.

The program is **not** required to handle invalid user input. Assume the license plate and owner names are provided in the expected formats.

## Design and Implementation Requirements

As you have learnt, decomposition in computer programming is breaking a complex problem or system into parts that are easier to conceive, understand, program, and maintain. Once you break down the problem into several parts/methods, you can tackle each part individually. Your solution MUST use functional decomposition.

Your finished program must follow all course coding standards provided by your instructor. In addition, you must follow the requirements below.

In the computation section, your program will need to produce several intermediate results and then assemble them into the final result values. Therefore, your program must introduce a variety of well-named, self-documenting variables for holding each of the intermediate and final data.

It is not necessary to store the entire registration number in one integer variable. In fact, due to the range limitations on values of the `int` type, this is not possible. Instead, generate individual digits and/or sub-sequences of digits. Then, assemble these to produce the final registration number. This number, as produced by the end of the "computation" section, must be represented as a correctly formatted string (i.e. with exactly 12 digit characters).

## Important Submission Instructions

Failure to follow these instructions will result in mark reduction on the assignment.

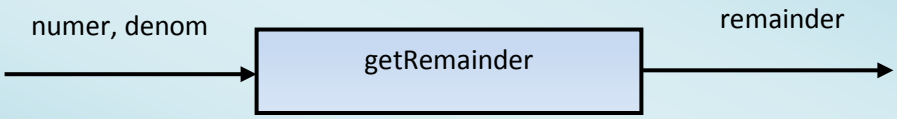
**Part 1: Design:**

You will create a single Word document that contains the following:

- IPO diagram for **each** method identified in the solution.
- The "contract" - from the method "recipe" done in lab previously which includes,
  - Name of the method
  - Method description
  - Inputs to the method: formal parameters (data types are not to be included)
  - Output of the method: return values, if any (data types are not to be included)

For example, suppose your problem had a method which computes remainders from division and returns the result back (7 mod 3, the resulting remainder is 1), the Part 1(Design) for that method would be as follows:

**IPO diagram:**




**The function/method "contract":**

| | |
|---|---|
| Name of the method: | getRemainder |
| Method Description: | This method computes the remainder from division and gives the result back. |
| Inputs to the Method: | the numerator, the denominator |
| Outputs of the method: | the remainder from division |

**You will do this for ALL methods within your program.**

Submit a MS Word file (`*.docx`) for this part. Name the file

`<lastName>_<firstName>_Asg2_Part1.docx`

**Part 2: Implementation:**

Submit a **BlueJ project folder** (with all its contents) to the "submit" folder, which appears under the "I:" drive each time you log in to a university PC. This folder must include a ".java" file containing an executable class with a main method. If you are submitting from a non-university computer (e.g. a home PC), you can access the submit folder via `secure.mtroyal.ca`, into which you can upload a compressed (e.g. ".zip") version of your ENTIRE folder.

Your submitted folder name must follow the format illustrated below:

```
<LastName>_<FirstName>_Asg2_Part2
```

### Documentation Template:

Replace all highlighted text with actual values

#### File header

```
/**
 *  <include description of the class here>
 * @author <your name>
 * @version 1.0
 * Last Modified: <date> - <action> <who made the change>
 */
```

### Background: Working with Dates in Java

Java provides a `GregorianCalendar` class for representing calendar dates. By default, when a newly created instance is initialized it is set to the current date and time (i.e. at the time the code was executed).

```
GregorianCalendar date = new GregorianCalendar(); // object rep's today
```

The properties of the date object can be queried as illustrated below:

```
System.out.println( "current year  = " + date.get(Calendar.YEAR) );
System.out.println( "current month = " + date.get(Calendar.MONTH) );
```

Similarly, its properties can be modified. For example:

```
date.set(Calendar.YEAR, 2025); // object now rep's a date in the future
```

A `GregorianCalendar` object is actually quite complicated, with several properties and sophisticated methods. To simplify the display of dates as text, Java also provides a `SimpleDateFormat` class for building strings, with a specified layout, from calendar objects. For example:

```
SimpleDateFormat dateFormatter = new SimpleDateFormat("dd/mm/yyyy");
System.out.println( dateFormatter.format(date.getTime()) );
```

The above code extracts the relevant information from the date object. The `dateFormatter` object is then told to format this information as text according to its pattern property (specified when the formatter was created).

To use these classes, the following imports are required:

```
import java.util.*;
import java.text.SimpleDateFormat;
```

The official Java documentation, searchable via the web, includes various tutorials which provide further information about using these classes.

### Grading

The tentative marking rubric to be used for grading will be posted soon.