PRGM10: Age Utilities

due: Sunday 10/30/16, 11:00 pm (42 pts)

Overview:

For this program, you're going to do several interesting things in the course of learning about **selection logic ("logical forks in the road")** in Java programs:

- **You'll begin creating a dedicated *Utils* class filled with useful, reusable static utilities**. Here, the first ones will be:
  - **A method to read one *int* value, using either a command-line *Scanner* or a GUI popup dialog.** After this, no more direct set-up of Scanner ever again – this is code reuse!
  - **A method to calculate an age accurately,** as of any reference date.
- **You'll develop a first short algorithm.** This one will require you to work with two input *CS12Date* objects, and determine an age, taking into account birthdays in the future, etc.
- **You'll exercise these first two utility methods using a short client class**. Calculate your own age as of today, obtaining inputs using both data input modes. Then see if your algorithm accurately predicts your own age as of some future "milestone" dates, such as your 50th birthday (or maybe later than that, for some of us ;-)
- **You'll get the beginnings of some ongoing exposure to creating some code from a program specification.** Synthesize a list of requirements, method interfaces, program structure, and some pseudocode… into the working code that is envisioned by the "customer".
- **You'll practice thoroughly testing your own code for accuracy.** Here, you will be given a test program against which to run your ***Utils*** class. It will automatically check your age calculations for a wide range of birthdates: past, present AND future. If any results are off, you will need to go back and debug your age logic algorithms!

We're going to be enhancing and adding to this *Utils* class, week by week, for the remainder of the semester, so it's critical that you get it correctly started in THIS assignment. The good news is, once we get one small capability just how we want it – it will be easy to "clone" it for other uses. For example, this week and next, we'll get a general purpose method in place for reading *int* values with certain error checking, so after that it will be trivial to clone it for reading doubles, chars, Strings, etc.

You will be given a starting shell for the utilities file, so just adapt it according to the program spec.

Java Objectives:

1) Selection logic (if-else, nested and/or compound) [CSLO-1]
2) Static methods [CSLO-1]
3) Method creation [CSLO-3]
4) Code modification and reuse [CSLO-3]
5) Use of existing classes Scanner, JOptionPane, and CS12Date [CSLO-2]
6) Translation of a problem specification in Java code [CSLO-4]
7) Application of Java to solve simple calculations [CSLO-4]

Preliminaries:

1) As always, begin by FIRST reviewing all the weekly lecture materials, videos, and source code examples. They should answer most, if not all, questions.

2) You'll also want to take another look back at the concept of **boolean flag variables**, in last week's "condition examples" notes. One of the mods you'll need to make will be very similar.

3) Start out by collecting into one place all the program raw materials you'll need:
   - This **program specification**
   - The starting utilities **code template**: **UtilsFL.java**
   - The **test driver code**, which you'll run to test your age algorithm: **UtilsTest1RL.java**
   - The ***CS12Date*** class: **CS12Date.java**
   - You will probably also find the ***CS12Date*** **API** useful here **CS12Date API and UML.pdf**

4) Rename the file **UtilsFL.java** to your own initials. Be sure to update the header block as needed. Inspect the layout of the file, and locate the 2 methods containing the text "TODO". **These are the <u>only</u> 2 methods in this file which will require updating.** See the **Requirements** section for the modifications needed.

5) In the test driver file **UtilsTest1RL.java**, do a global replace of the pattern "**UtilsRL**" to the name of YOUR utilities file. Within jGRASP, use the Edit: Find/Replace menu item, or the Ctrl-F shortcut.

6) Then, compile and execute the file **UtilsTest1RL** and make sure it runs to completion. **Most of the test cases will fail with an age of -1**, but now you have some working test code against which to test <u>your</u> age algorithm, as you develop it.

7) Begin developing the needed age algorithm in your **UtilsFL.java** file. See the requirements and the various design descriptions, to get you pointed in the right direction.

8) Finally, begin by creating the client file **AgeClientFL.java**. Use your template as a starting point. Don't forget to update the comment block accordingly.

9) Observe the same general guidelines as for all programs:
   - Declare any needed variables and objects <u>upfront</u> in your program. Use good variable names, and observe all **camelCase** or **DECLARED_CONSTANT** naming conventions.
   - Comment the sections of your code briefly as you write it, explain what's going on.
   - Use some whitespace in your outputs, don't cram everything together. Clearly label all outputs.

   By this point in the course, good general adherence to the course Coding Standard is expected on all programs!

Requirements:

Write a program which meets the following requirements:

1) **Structure**: Your application must consist of two Java classes:
    a. An ***AgeClientFL*** client class (created from your program template)
    b. A ***UtilsFL*** utilities file class (created from the provided starter code)
    c. You will also need to use the ***CS12Date*** class
    d. You should also use the provided ***UtilsTest1RL*** test code to check your program
    e. See the **Structure** section which follows for the expected code structure.

2) **Utility methods**: Your utilities class needs the following modifications:
    a. See the **Design** section which follows for the expected method interfaces.
    b. Modify only the indicated "TODO" code. **All other methods are to be left unaltered**.
    c. All methods in this class are to be called statically: **UtilsFL.method()**

    d. In the **readInt()** method:
        i. Add a 2nd boolean input "mode flag" to allow switching between data input sources:
            1. Use the common data input prompt text for both input modes.
            2. If the mode flag is *false*, use the provided ***Scanner*** input code.
            3. If the mode flag is *true*, add the equivalent ***JOptionPane*** input code.
        ii. Use appropriate selection logic.
        iii. For either input mode, simply return one common *int* value to the caller (**do NOT print anything from this method**).

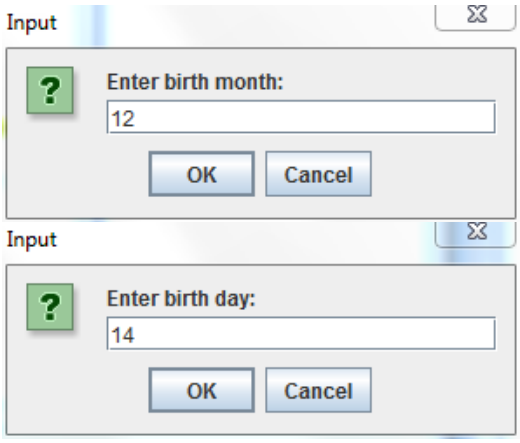
    e. In the **2-input *getAge*() ** method:
        i. Create an algorithm which correctly calculates the current age for a given birthdate, "as of" any reference date (you should assume that the reference date in general is NOT the current date!)
            1. The 1-input version of ***getAge()*** internally calls the 2-input version of ***getAge()***, with the 2nd input provided as the current date, but needs no modification.
        ii. Return the calculated age to the client, without printing it – just return the age.
        iii. If the birthdate is AFTER the reference date in time, return a -1 value to the client, and **print an error message to the command line from *getAge*().**
        iv. **The *getAge()* method does not need to do any input error checking**. Assume that impermissible dates such as 4/31/2000, 13/12/2008, or 2/29/2015 are intercepted by the upstream creation of CS12Date input objects.
        v. See the **Age Algorithm** notes which follow, for more details on how to proceed.

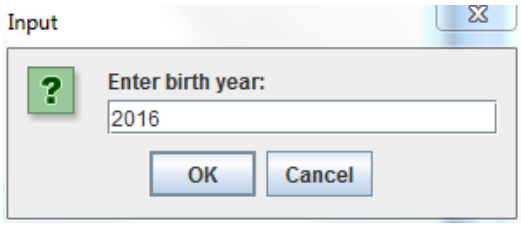
3) **Client program**: Your client class ***AgeClientFL*** needs to exercise the utilities class, by doing the following 3 things:
    a. First, prompt the user 3 times for month, day, and year using the ***Scanner*** (false) mode of ***readInt()***.
        i. Assemble these into a CS12Date object, use this to call the **1-input form of *getAge()***, and display the input date AND the resultant age, to the command line.
    b. Second, prompt the user 3 times for month, day, and year using the ***JOptionPane*** (true) mode of ***readInt()***.
        i. Assemble these into a CS12Date object, use this to call the **1-input form of *getAge()***, and display the input date AND the resultant age, to the command line.
    c. Third, create two new *CS12Date* objects (can hardwire these, no need to prompt for them). One should be <u>your</u> birthdate, the other some future "milestone" birthdate, such as your 21st or 50th BD, perhaps.
        i. Use both these dates to call the **2-input form of *getAge()*,** and **display <u>both</u> dates <u>and</u> the calculated age as of that reference date.**
    d. **There should be NO direct use of *Scanner* or *JOptionPane* within your client *main()* method**, use your created ***readInt()*** Utils method for input instead.
    e. See the sample outputs which follow for examples of the expected client code output.

Sample output:

Your resulting client program outputs should look something like this (you can label it however you wish). Here, we are exercising both the input modes of our ***UtilsFL.readInt()*** method:

```
----jGRASP exec: java AgeClientRL
Enter birth month: 11
Enter birth day: 9
Enter birth year: 2001
birthday: 11/9/2001     age = 14
```





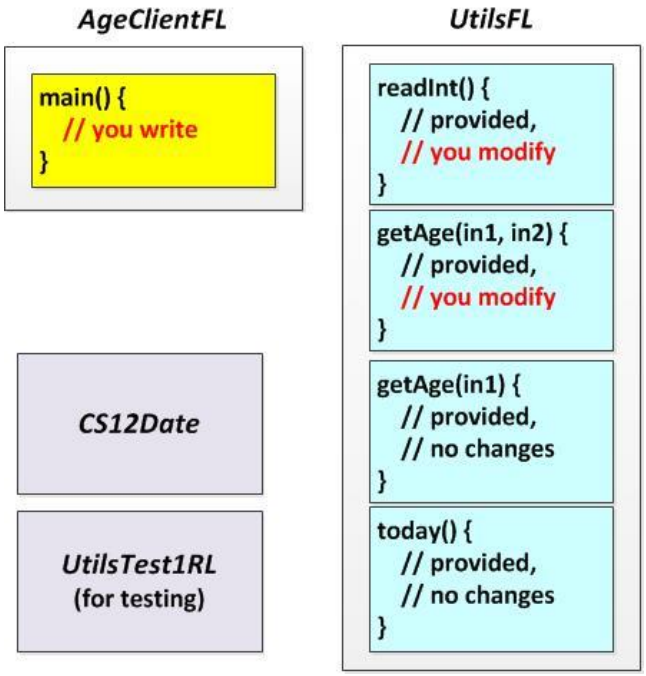

```
(from GUI inputs)
ERROR: provided birthdate 12/14/2016 is after current date
birthday: 12/14/2016    age = -1

birthday: 7/18/1963     age = 65        as of: 7/18/2028

----jGRASP: operation complete.
```

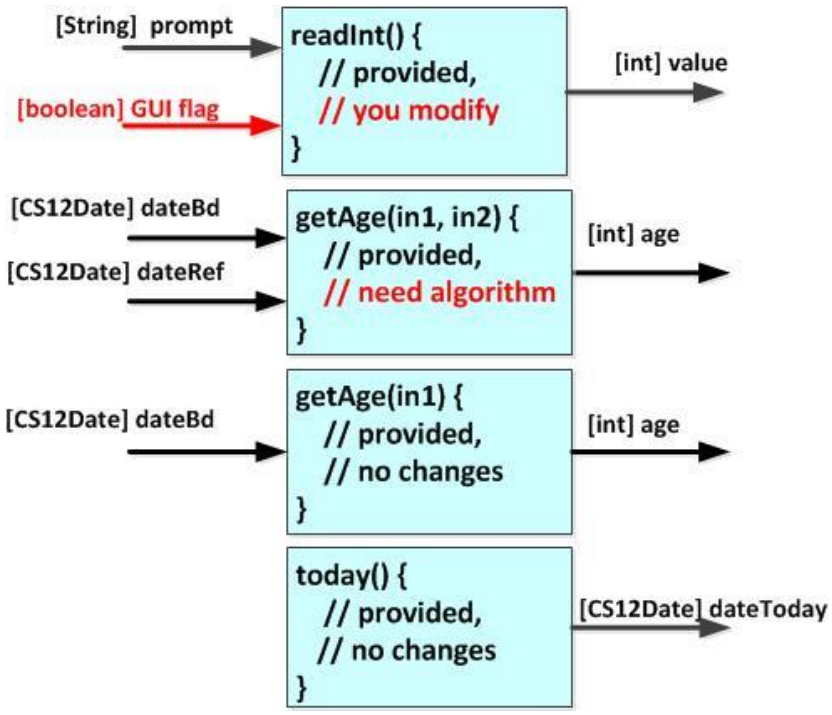
Structure:

These are the primary 2 Java classes needed for your application, plus 2 more in a supporting capacity.



Design:

These are the interfaces for the required ***UtilsFL*** class. Changes you need to make are indicated in **RED**.



Age Algorithm:

There are two dates to consider: a **birthdate** and a **reference date**. The reference date might be (but doesn't have to be) today's date, in which case the age is "as of" today. The reference date could also be some past or future date, in which case the age is "as of" that particular date.

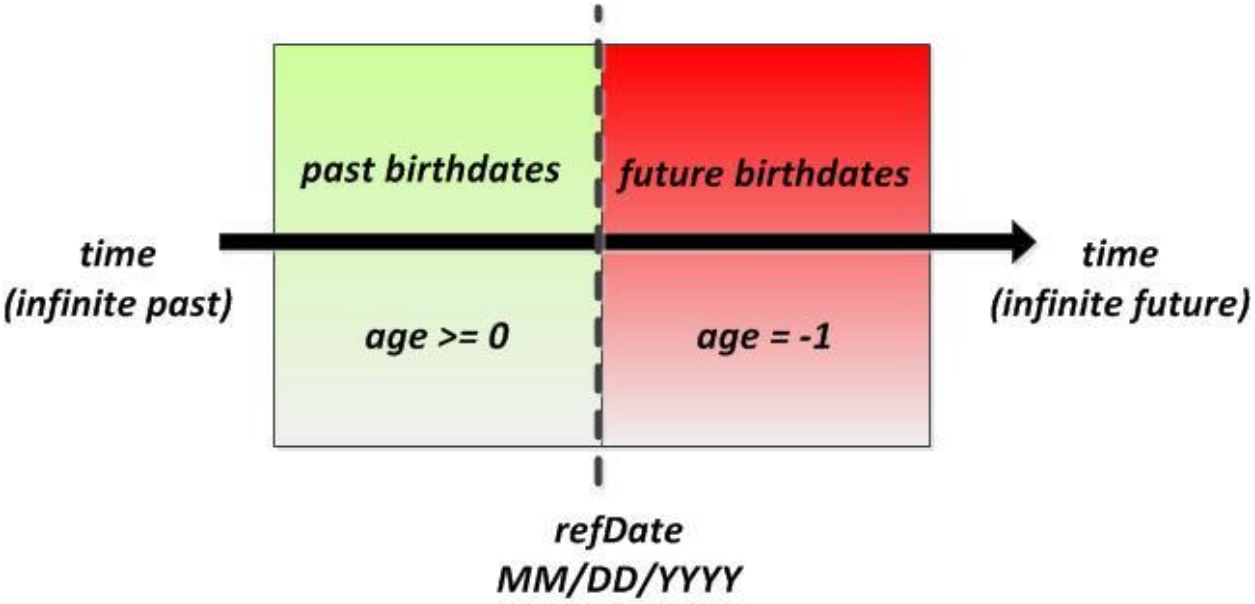
Think in terms of the reference date, not simply today's date. If the reference date "happens" to be today, you'll get the age as of today. But if not, you'll get the more general result of an age as of any date. **You had an age "as of" your high-school graduation day, and you will have an age "as of" the day before your 65th birthday.**

As long as the birthdate is BEFORE the reference date in time, the age algorithm should correctly return the age in whole-numbered years "as of" that reference date. Some examples:

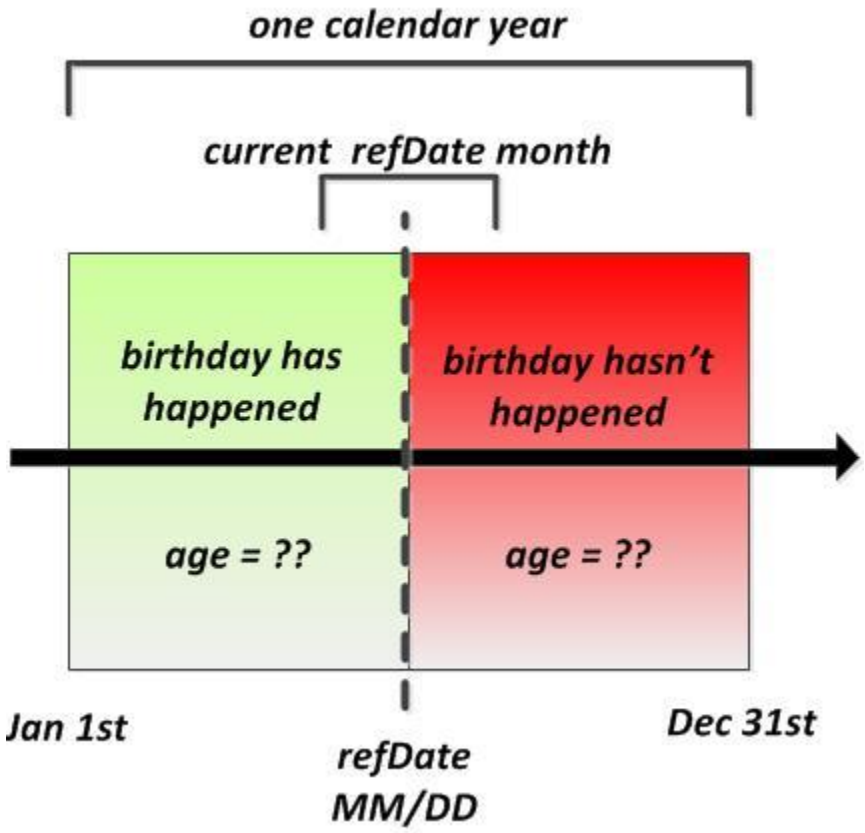
- A newborn who hasn't yet celebrated their first BD should have an age of 0.
- Some notable (dead) person might have an age that's hundreds of years old, "as of" a certain reference date. George Washington should have an age "as of" today, yesterday, or any other date.
- YOU should also have a correct age "as of" your 5th or 18th or 65th or 80th BD. Or even as of today!
- If your own BD in the calendar year hasn't happened yet, your age must still be correct. For example, if your BD is in December, your age right now is one less than the usual difference of years might indicate.

To tackle the age problem, first determine:

- Is the birthdate in the future "in time" from the refDate? If so, the age is -1 by our definition.
- Otherwise, the person's birthdate is either on the refDate or at some time in the past, and so has a legitimate age. See the next diagram for how to think about THAT situation.



THEN, if you know that the birthdate has occurred "in time", keep checking for the age. Specifically, **where within one calendar year does the birthdate fall?** Has that birthdate taken place or not in the calendar year?? See the diagram below.



Pseudocode (one possible approach):

- Take apart the birthdate and reference date (2 CS12Date objects) into 2 month, day, year triples (for a total of 6 individual ints - how do you do this??). Name them clearly!
- Determine whether the birthdate is in the future from the reference date.
    - Is it a future year? A future month this year? A future day this month?
    - How do you express such logic programmatically?
- If the birthdate is NOT in the future from the reference date, determine the real age
    - What's the simplistic, first-cut "nominal" age?
    - On which side of the above dividing line in the year does the birthday fall?
    - What's the age if the birthdate is in a future month? In a past month?
    - What's the age if the birthdate is earlier or later in the current month?
    - How do you express the above using logical comparisons?
- What kind of selection structure makes the most sense to YOU?
    - Multiple branches of if-elseif-else?
    - Nested if-else structures?
- Take a look at the available ***CS12Date* API methods**. Is there anything useful in there for this problem?

Suggested Implementation:

I suggest the following sequence of implementation. Take things one small step at a time!

- **First, make sure you understand how the provided starter *Utils* class works**. Use the existing ***readInt()*** to prompt for 3 int values in your client program, use these to construct 2 *CS12Date* objects, and use these objects to call both the ***getAge()*** methods. Print the resulting ages, which should be -1 for now, but that's OK as a starting point.
- **Next, update the *readInt*() in your *Utils* class.** Add the second input flag, implement GUI popup input, and test both input modes from your client program, to set up your *CS12Date* objects.
- **Next, implement the age algorithm.** Develop the algorithm in stages. For example, first figure out how to reject future dates (later than the reference date) and return a -1 age. Then, consider how to express birthdates within one calendar year that are BEFORE and AFTER the reference date. See the preceding age algorithm notes.
- **Test your program.** Check your results against the provided sample outputs. See the comments below.

Testing:

- **Be sure to test your code before submitting it**. Each execution should prompt for two birthdates, each having separate month, day and year components: **once from the command line, once from GUI popups**. Echo the provided date and the display the corresponding age for each birthdate. For the milestone case, also print the "as of" date. ***See the sample outputs for what you output should look like.***
- **Run your program several times.** Test your code with a handful of dates, both before and after (and ON) the current date. Test your program with past, present, and future dates. Make sure that invalid birthdates (those after the current date) generate an error message from within ***getAge()***. But, your submitted program only needs to prompt for two dates (one via command line, one via GUI popup).
- **Test your own program versus the provided test driver program**. Instructions for running it are given in **Preliminaries** and within the file itself. **Make sure that all test results pass**. If any don't pass, go back and tighten up your age algorithm! ***I will run your submitted program against this test driver for numerical correctness, so you MUST adhere to the method interfaces described in this document and the starter UtilsFL code.***
- **Clean up your final code.** Check your program for good layout, complete header, code indentations, braces usage, and logic commenting (each method needs to have a one-liner, and there should be reasonable internal commenting also). These things will begin to receive more emphasis, especially now that the logic becomes more deeply nested. ***Check your code against the grading rubric before submitting it, to make sure you haven't overlooked anything.***
- **Diagnostic outputs**: It's OK (and in fact, encouraged) to include plenty of test code and printlns while developing your logic. You may leave this code in place in your program (to a reasonable amount) by simply commenting it out. However, make sure any residual printouts are suppressed (commented out) in your final submittal. **I do NOT want to see all kinds of diagnostic outputs!**

Sample Test Driver Outputs:

Running the provided test code driver should look something like this. **Look for all "pass" results**. If not, go back and debug your age logic!

```
 ----jGRASP exec: java UtilsTest1RL
============================
      Age test results
(test driver client provided)
============================
Birthdays BEFORE current date:
bd:  10/19/1986   as of: 10/20/2016    age:  30  expected:  30  pass: true
bd:  10/19/2016   as of: 10/20/2016    age:   0  expected:   0  pass: true
bd:    3/1/1981   as of: 10/20/2016    age:  35  expected:  35  pass: true
bd:    2/9/2016   as of: 10/20/2016    age:   0  expected:   0  pass: true
bd:   9/28/1989   as of: 10/20/2016    age:  27  expected:  27  pass: true
bd:   2/27/2016   as of: 10/20/2016    age:   0  expected:   0  pass: true
bd:   10/1/1934   as of: 10/20/2016    age:  82  expected:  82  pass: true
bd:   10/1/2016   as of: 10/20/2016    age:   0  expected:   0  pass: true

Birthdays AFTER current date:
bd:  10/21/1969   as of: 10/20/2016    age:  46  expected:  46  pass: true
bd:   12/9/2000   as of: 10/20/2016    age:  15  expected:  15  pass: true
bd:   12/4/1984   as of: 10/20/2016    age:  31  expected:  31  pass: true
bd:  12/30/1995   as of: 10/20/2016    age:  20  expected:  20  pass: true
bd:  12/29/2013   as of: 10/20/2016    age:   2  expected:   2  pass: true

Birthdays ON current date:
bd:  10/20/2016   as of: 10/20/2016    age:   0  expected:   0  pass: true
bd:  10/20/2008   as of: 10/20/2016    age:   8  expected:   8  pass: true
bd:  10/20/1994   as of: 10/20/2016    age:  22  expected:  22  pass: true
bd:  10/20/1964   as of: 10/20/2016    age:  52  expected:  52  pass: true
bd:  10/20/1944   as of: 10/20/2016    age:  72  expected:  72  pass: true

Birthdays in FUTURE:
==> error message to user **generated by getAge()** must display for EACH future date

ERROR: provided birthdate 10/21/2016 is after current date
bd:  10/21/2016   as of: 10/20/2016    age:  -1  expected:  -1  pass: true
ERROR: provided birthdate 10/30/2016 is after current date
bd:  10/30/2016   as of: 10/20/2016    age:  -1  expected:  -1  pass: true
ERROR: provided birthdate 12/1/2016 is after current date
bd:   12/1/2016   as of: 10/20/2016    age:  -1  expected:  -1  pass: true
ERROR: provided birthdate 12/28/2016 is after current date
bd:  12/28/2016   as of: 10/20/2016    age:  -1  expected:  -1  pass: true
ERROR: provided birthdate 2/15/2017 is after current date
bd:   2/15/2017   as of: 10/20/2016    age:  -1  expected:  -1  pass: true
ERROR: provided birthdate 11/9/2018 is after current date
bd:   11/9/2018   as of: 10/20/2016    age:  -1  expected:  -1  pass: true

Prior birthdays (dead or alive) on future milestone dates:
bd:   2/22/1732      on:  2/22/2032    age: 300  expected: 300  pass: true    George Washington 300th
bd:   2/22/1732      on:  2/21/2032    age: 299  expected: 299  pass: true    George Washington 300th, day before
bd:   7/18/1963      on:  7/18/2028    age:  65  expected:  65  pass: true    instructor 65th
bd:   7/18/1963      on:  7/17/2028    age:  64  expected:  64  pass: true    instructor 65th, day before

 ----jGRASP: operation complete.
```

Submitting the assignment:

Turn in the 2 Java files called **AgeClientFL.java** and **UtilsFL.java**

Submit the both Java source files in Canvas, under this assignment. You don't need to also include ***CS12Date*** or the test driver file.

Make sure your code adheres to all the usual coding conventions described in the course Coding Standard. **Check your program against the rubric below before submitting it!**

Grading:

See the rubric associated with this assignment in Canvas, or below.

	PRGM10: TOTAL POINTS POSSIBLE:	**42**
1.00	**Program Elements/Structure**	**20**
1.01	2-file solution: Utils and AgeClient classes	1
1.02	Use of provided Utils template, other methods left unaltered	2
1.03	Utils readInt() with proper input flag, 2 input modes, proper mode convention	3
1.04	Utils getAge() 2-input version: age algorithm, return int value WITHOUT printing	4
1.05	Utils getAge() 2-input version: error checking, age= -1 returned WITH message printed	2
1.06	AgeClient, prompt for 3 non-GUI inputs, 1-input call to calculate age	2
1.07	AgeClient, prompt for 3 GUI inputs, 1-input call to calculate age	2
1.08	AgeClient, milestone BD and 2-input call to calculate age (can hardwire dates)	2
1.09	NO Scanner, GUI, age logic, or error message code directly inside client (static util calls only)	2
2.00	**Program Execution**	**15**
2.01	Both files compile and run as-submitted	2
2.02	Adherence to specified interface, test program compiles/runs against Utils	2
2.03	Correct age results for all test program cases	5
2.04	All outputs clearly displayed and labeled, date(s) AND age for each output case (x3)	3
2.05	Correct age results for all 3 client cases	3
3.00	**Program Style**	**7**
3.01	General adherence to coding standard	5
3.02	Adequate commenting of all program logic, methods	2
4.00	**Late Deduction**	**0**
4.01	3 pts/day	0