

PRGM19: Age Utils

due: Sunday 04/19/15, 11:00 pm (40 pts)

Overview:

For this program, you will work with if-else selection logic. You will also begin putting together a class of useful (static) standalone utilities. These reusable utilities will be added to, and used over and over again, on other programs this semester.

To complete this assignment, you need to do 3 things:

- **MODIFY** one existing utility method which returns an *int*, so it has command line AND popup versions
- **WRITE** one new utility method to calculate some ages, given 2 *SimpleDates*: a birthdate and a reference date
- **TEST** your methods, by using them inside your client class `main()` method

Recall that static methods are useful for one-time, standalone capabilities, without needing to create and maintain objects. Static methods simply add the ***static*** keyword to their method interface. You will be provided with a starting example file, which you will then adapt and add to for this assignment.

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 `SimpleDate` [CSLO-2]
- 6) Application of Java to solve simple calculations [CSLO-4]

Requirements:

Write a program which meets the following requirements:

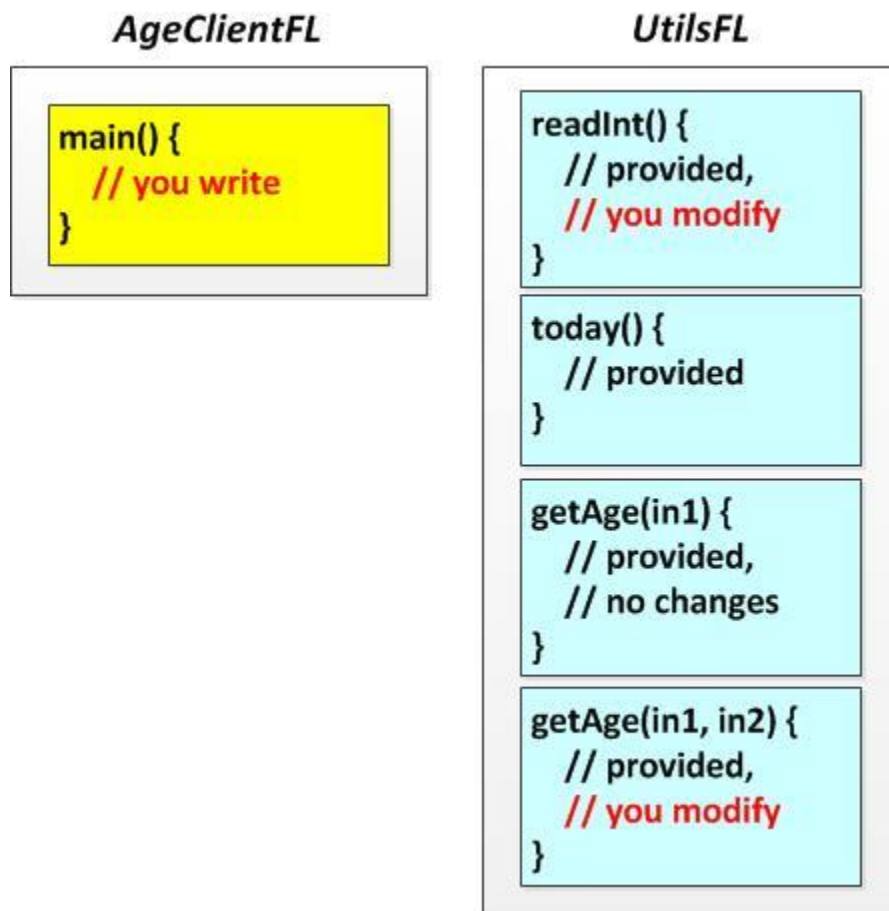
1. Your application must consist of two Java classes: an *AgeClientFL* client class, and a *UtilsFL* utilities file class. You will also need to use the existing *SimpleDate* class. See the [Structure section which follows](#).
2. Starter code for your *UtilsFL* class is provided. You must make modifications to do the following. See the **Design** section which follows.
 - a. In **readInt()**, add a GUI option which allows for data input from *JOptionPane* popups. Add an input argument flag and program logic to use *Scanner* if false, *JOptionPane* if true. There should be two if-else cases for these.
 - b. In the 2-input **getAge()**, create an algorithm which correctly calculates the current age, given a birthdate and a reference date. If the birthdate is AFTER the reference date, return a -1 to the caller, and print an error message to the command line.
 - c. The existing methods for **today()** and the 1-input **getAge()** are not to be modified.
 - d. All the methods in the utilities file are to be called statically: **UtilsFL.method()**
3. Your client application will be written in the main() method of **AgeClientFL**, and needs to do the following:
 - a. Prompt the user 3 times for month, day, and year using the non-GUI (false) mode of readInt().
 - b. Prompt the user 3 times for month, day, and year using the GUI (true) mode of readInt().
 - c. Assemble each triplet of data into 2 SimpleDate objects, and use these to calculate the age as of the current date. Use the single-input form of readInt().
 - d. Display each SimpleDate object and the corresponding age as of the current date. If the birthdate is AFTER the current date, an age of -1 should be displayed, and a command line message displayed to the user.
 - e. As a final check, create a 3rd SimpleDate object which corresponds to some “milestone” future birthday of yours (21st or 40th or 65th BD, perhaps), and use the 2-input form of readInt() to calculate and display your age on that day. You do NOT have to prompt the user for these dates, you may hardcode them into your program. Display both BD and as-of dates, and the resulting age.
 - f. Your client application must NOT use Scanner or JOptionPane directly, nor calculate the ages directly. Only use the (static) methods of your utilities file for this purpose.
4. **Test your age algorithm with enough date cases to confirm it is giving correct results for ANY reference date. Consider past and future days, months, years, as well as today’s date. I will run your UtilsFL class against a test program to check for correct age calculations!**

Structure:

- Your program will be contained within two Java files: a client application class **AgeClientFL**, and a utility methods file **UtilsFL**.
- All of the methods in the utilities file will be STATIC methods. This simply means they will contain a static keyword in their interface, and they are each called from the external client as:

UtilsFL.method();

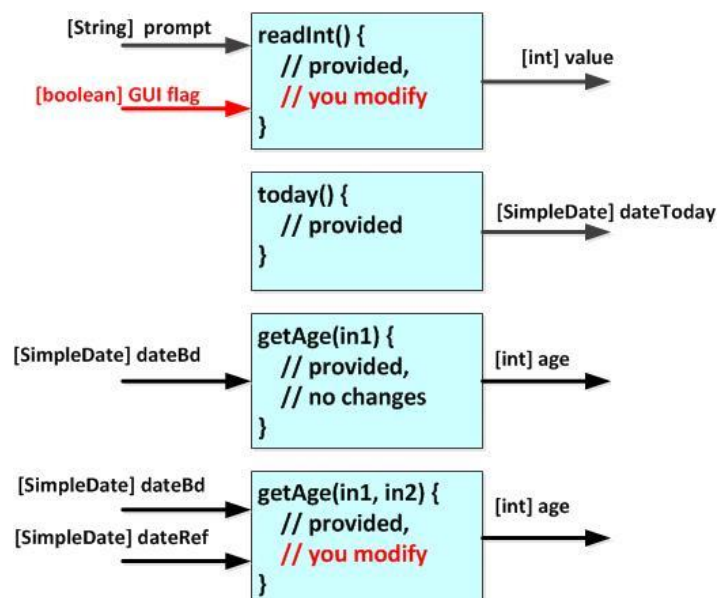
- A starter file for UtilsFL is provided. You need to rename this file to add your own initials, and then begin modifying it as described on the next page.
- **Your main() method should use the adapted readInt() method to obtain all user inputs. Do not use Scanner or JOptionPane directly inside your main().**
- The overall structure of your program should be something like this:



Design:

Your **UtilsFL** file should contain the following methods. The method interfaces are shown below.

- **A method to read an int using *Scanner*, given a *String* prompt** (this is what you already did on the Methods HW). You need to adapt this method such that, given a *String* input prompt, it reads input data using *Scanner* OR *JOptionPane*, depending upon a GUI flag.
 - Do NOT add a new method. Instead, adapt the existing method by adding a second input argument and some if-else logic.
 - Once you have completed this, it will be trivial to clone this method later to handle *doubles*, *chars*, or any other datatype.
- **A method which returns the current date as a *SimpleDate*.** This is provided. No mods are needed to this. Test it for yourself, to be clear on how it works.
- **A method to calculate age.** The first one has only one input argument, and calls the other method in nested fashion using *today()* as the second argument. No mods are needed here.
- **A second method to calculate age.** You need to create an algorithm which calculates the correct age, given a birthdate AND a reference as-of date.
 - Start by taking apart both *SimpleDates* internally, into 6 individual month/day/year variables. With these, you have all the needed information to correctly calculate the age.
 - You will need to come up with a suitable algorithm for age. Take into account whether or not one's birthday in the current year has arrived yet.
 - Provide error checking on the input birthdate. If the input birthdate is AFTER the reference date, print a message to the user and return a -1 as the age.
 - You do not need to check for impermissible dates such as 4/31/2000, 13/12/2000, or 2/29/2014. Assume the *SimpleDate* class itself intercepts such things and takes some corrective actions.



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. Echo the provided date and the display the corresponding age for each birthdate.

Run your program several times. Test your code with a handful of dates, both before and after (and ON) the current date. Make sure that invalid birthdates (those after the current date) are noted as such. But, your submitted program only needs to prompt for two dates (one via command line, one via GUI popup).

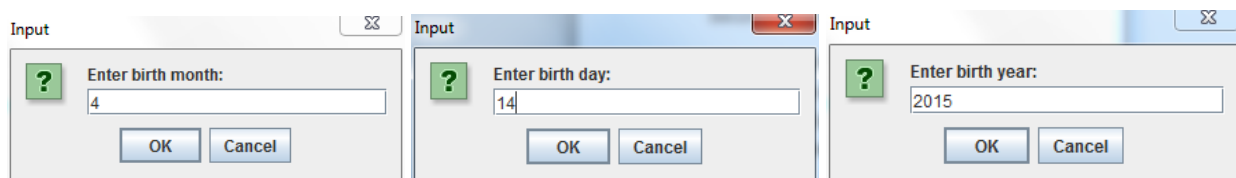
I will test your UtilsFL's numerical correctness by running it with a test driver program I will create, checking ages against multiple dates. So make sure you have tested your own code well first! As such, you MUST adhere to the method interfaces given in this document and in the starter UtilsFL code.

Check your program for good layout, complete headers, code indentations, braces usage, and commenting (each method needs to have a one-liner, and there should be reasonable internal comments also). These things will begin to receive more emphasis, especially now that the logic becomes more deeply nested.

It's OK to include plenty of test code and printlns while developing your code, and you may leave this code in place in your program by commenting it out (to a reasonable amount). However, make sure any residual printouts are suppressed (commented out) in your final submittal, and do a general code cleanup before submitting.

Output:

Your output should look something like this (exact ordering may differ somewhat...)

Three small GUI windows titled 'Input' are shown side-by-side. Each window has a green question mark icon in a box on the left. The first window prompts 'Enter birth month:' with the value '4' entered in the text field. The second window prompts 'Enter birth day:' with the value '14' entered. The third window prompts 'Enter birth year:' with the value '2015' entered. Each window has 'OK' and 'Cancel' buttons at the bottom.

```
----jGRASP exec: java AgeClientRL
>> Enter birth month: 11
>> Enter birth day: 9
>> Enter birth year: 2001
birthday: 11/9/2001    age = 13

(GUI inputs)
ERROR: provided birthdate 4/14/2015 is after current date
birthday: 4/14/2015    age = -1

birthday: 7/18/1963    age = 65    as of: 7/18/2028
----jGRASP: operation complete.
```

Submitting the assignment:

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

Submit both Java source files in Canvas, under this assignment. I don't need to also receive SimpleDate.java, I have that one already.

Grading:

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

PRGM19: TOTAL POINTS POSSIBLE:		40
1.00	Program Elements	16
1.01	Main client class: uses util methods, obtains M/D/Y, calls age and displays it (x2)	4
1.02	Util class readInt(): modified for added parameter, added option for GUI inputs	2
1.03	Util class today(): used as-provided, no mods	1
1.04	Util class getAge(): 1-arg version used as-provided, no mods	1
1.05	Util class getAge(): 2-arg version interface is maintained w/o mods	1
1.06	Util class getAge(): age algorithm implemented correctly	5
1.07	Util class getAge(): error checking on future birthdate, -1 return value and message	2
2.00	Program Execution	15
2.01	Compiles and runs as submitted	2
2.02	Program prompts for command line inputs (x3) and popup GUI inputs (x3)	2
2.03	Both birthdate and age are displayed for both inputs	2
2.04	Error age and message for birthdates later in time than today	2
2.05	Birthdate, future milestone date, and correct age displayed (no prompts)	2
2.06	Correct ages displayed running test driver program	5
3.00	Program Style	9
3.01	Header block complete	1
3.02	Consistent nesting, braces	2
3.03	General layout, whitespace, readability	2
3.04	Sufficient commenting within program and for EACH method	2
3.05	Code cleanup done, stray outputs and dead code removed	2
4.00	Late Deduction	0
4.01	3 pts/day, prorated	0