Program #3 - Maybree the Menu Builder

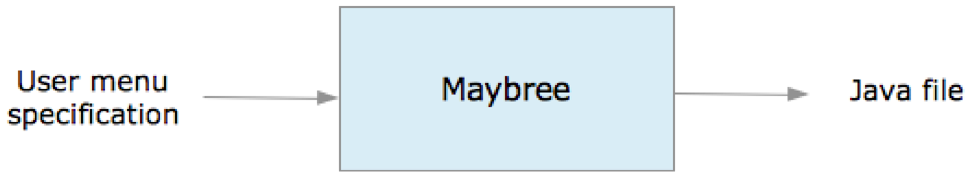
Maybree creates Java code that builds a menu

Logistics:

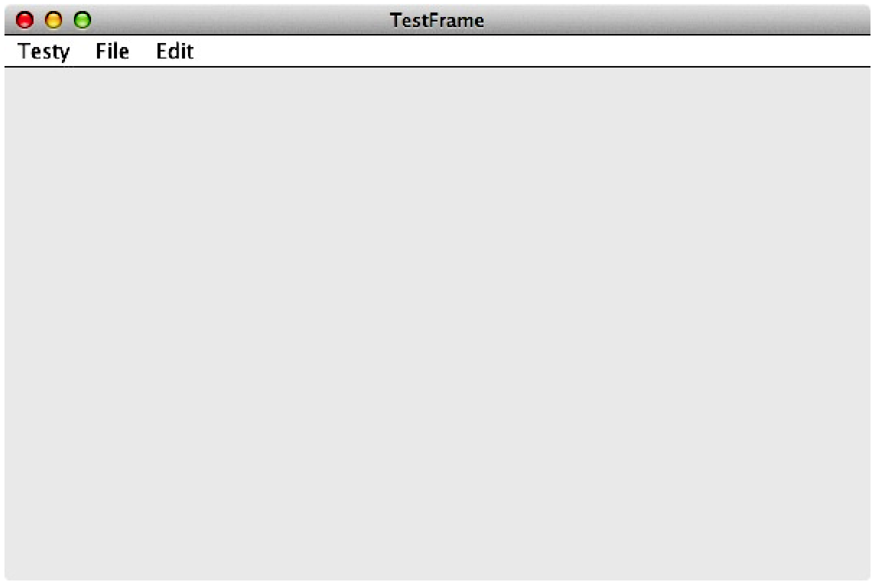
- Due: **Wed Feb 25, 2015**
- Worth: **6 points** (6% of your class grade)

1. Description

The input to Program #3 is the user's specification of a menu system. The output is a Java class that implements that menu. Like this:



For example, if the user specifies that a class name `TestFrame` be created with menus Testy, File, and Edit, then running the Java file created by Maybree will look like this:



2. Implementation

Well, Program #3 has two primary design issues: input and output.
Maybree is a console program that asks for the input file and then rolls.

2.1. Input file

We need to design a file that allows the user to specify his menu system and the options he wants to control the Java Maybree produces. Maybe something like this?

```
# This is a comment.
ClassName TestFrame
ActionListener TestListener
IncludeMain true

Menu Testy About Quit
Menu File Open Save Close
Menu Edit Copy Cut Paste
```

About this file:

- We have the standard # comment character.
- Blank lines are ignored.
- The first string in a line is (sort of) the command, then the arguments.
- Some commands are simple name, then value: ClassName, ActionListener, and IncludeMain.
- Menu is a complex command. The first string is the menu name. The remaining strings (if any) are menu items.

[Am I missing anything?]

All commands are optional. The commands and their default values are:

- ➔ **ClassName** - the name of the class we're creating (and the name of our file will be ClassName.java). Default: MaybreeExample.
- ➔ **ActionListener** - the name of the private class we'll use as ActionListener. Default: MaybreeListener.
- ➔ **IncludeMain** - if true, then a tiny main() is added to the bottom. If false, then no main(). Default: false.
- ➔ **Menu** - specifies a menu and its items. Obviously, there can be multiple Menu commands in a file. Default: Maybree menu with one menu item, Exit.

2.2. Output file

We need to design the structure of the Java files we write. The best way to do this is to code up a simple menu and understand the structure of the Java file. We'll talk about this in class.

Details:

- Ideally, the user's choices will be located in one spot (variables in the ctor?) and the rest of the code Maybree creates will use those choices.
- You need a `JFrame` to have a menu. You can design your class using either inheritance (is-a `JFrame)` or composition (has-a `JFrame)`. Your choice.
- Ideally (again), you won't need a method for each menu you create: buildTestyMenu(), buildFileMenu(), buildEditMenu(). Perhaps, you can create 1 method for all three that accepts as parameters the menu name and its items.
- Your listener should just print the menu item chosen.
- Of course, your output file must compile and run without errors.

2.3. Test cases

Once we are totally set on the input file specification, then I'll create a test suite for you to run.
One of my test files will be empty. Since all commands are optional, what does a menu with all the defaults used look like?

3. Grading

Create a `program3` folder in your k: drive.
Place these files in that folder:

- A `README` file describing the state of your program.
- All the Java files that comprise your Program #3 solution
- A `test` folder of your Program #3 test results… the Java files you created
- **NEW -** I have 4 tests for you to run. Please create one interesting test yourself.

All your code must follow our 161 Coding Guidelines. Your code must be **beautiful**! Ugly code will be penalized with a 0-100% reduction in points. A program that doesn't even compile is worth 0 points.

Good luck with Maybree!
thanks… yow, bill

PS - Some Program #3 design notes++
Important - I will **not** be moving the deadline on Program #3 past Wednesday.
We can't do this because of Friday's impending Exam #2!

My Javadoc

I have posted the Javadoc of my solution here:
wtkrieger.faculty.noctrl.edu/csc161/program3/javadoc/
Your solution may differ.

Some psuedo-code

Here's the pseudo-code of my `main()` in `Program03:`

```
main() {
      ask user for input file name
      create Maybree
      read the user's input file
      write the java
}
```

In Maybree, reading the user's menu spec file might involve:

- I used a `BufferedReader` and `readLine()`. You might prefer using `Scanner.`
- I used `PrintWriter` and println() to write my output file.
- I split each input line into tokens using `String.split()`. The first token is the command, and so on.
- You might use `StringBuilder` to create large strings, one piece at a time.

Check Snippets for: `PrintWriter, String.split()` and `StringBuilder.`

If you create a target Java file with markers ($MAYB1, $MAYB2, etc) signifying lines that need to change, then your `writeJava()` pseudocode is something like this:

```
open input target file
open an output PrintWriter
for each input line {
      if( line.startsWith( "$MAYBREE1")) {
            write public class XXX to output file
      }
      else if( line.startsWith( "$MAYBREE2")) {
            write correct ctor to output file
      }
      . . .
      else {
            echo line to output file
      }
}
```