

## CS 235 - Homework 11

### Deadline:

Problem 1 - presentation: to be given **DURING LAB** on **Wednesday, December 9**.

All other parts: due by **11:59 pm** on **Wednesday, December 9, 2015**.

### How to submit:

Submit your files using `~st10/235submit` on nrs-projects, with a homework number of 11, by the deadline shown above.

### Purpose

To practice preparing and giving a short Java live-program-demonstration, to reflect on Java and the Java programming/concepts you have been practicing this semester, to provide practice with JUnit, to practice a bit of file output using `PrintWriter`, to practice a bit of file input using `Scanner`, and to practice a bit using an `ArrayList`.

### Important notes:

- Follow the Java coding standards mentioned in previous homework handouts and discussed in class.
- Note that Java applications with graphical user interfaces are expected to be structured in the way demonstrated in the in-class examples (as in `ButtonTest.java`)
  - ...but using appropriate additional helper methods in the `JPanel` subclass is fine, as is using additional classes. IF you use additional public classes, be sure to submit those as well!
- It is possible that some of your programs may be posted to the course Moodle site.

### Problem 1 - presentation/reflection - presentation to be given DURING LAB on Wednesday, December 9

To provide you with some practice preparing and giving a short Java live-program-demonstration, and to reflect on some of the Java programming/concepts you have been practicing this semester, you are to present a favorite Java program that you have written this semester.

**Note that this presentation will also be the Week 15 Lab Exercise.**

Your presentation must meet the following specifications:

- You should plan for it to be from **about 2-3 minutes** in length.
- This must be a **live-program-demonstration** -- you are expected to **actually run** your selected program as part of your demonstration, **projecting** your program's output/side-effects/results.
  - Bring a thumb drive to lab containing your program, so you can insert it into the instructor's workstation for hopefully(!)-smooth execution and display.

- (If bringing a thumb drive containing your program would be a problem, then E-MAIL me your program by 10:00 pm on TUESDAY, DECEMBER 8th and I'll put it on a drive and bring it to lab.)
- You are also expected to say, as part of the presentation:
  - **why** you selected your selected program for presentation
  - **what** you like about your selected program
- Your selected Java program should be something you have written this semester;
  - can it be something you wrote for one of the course homeworks? Yes!
  - can it be something that you wrote that was not part of a course homework? Yes!
  - can it be something you write between now and the Week 15 Lab, just for this presentation? Yes!
  - does it matter if more than one class member happens to choose their version of the same homework problems to display? No, that it NOT a problem.

By the homework deadline, also submit the `.java` source code file/files for your selected program.

## Problem 2

Consider Homework 3 - Problem 1's class `CookeryFan`. (Note that there's a posted example version of this on the course Moodle site, under "Selected solutions", if you would prefer using it for this problem instead of your version.)

Write a JUnit test class `CookeryFanTest` testing `CookeryFan`, using the style/guidelines discussed in class (and demonstrated in `GameDieTest.java`), and meeting at least the following additional requirements:

- declare at least one private `CookeryFan` data field (more are fine, but include at least one)
- include a method `setUp` that instantiates your `CookeryFan` data field(s)
- write an appropriate (and appropriately-named) test method for each `CookeryFan` method except for the constructor (we'll let the test methods for the getters/selector methods essentially also test the constructor).

Submit your resulting `CookeryFanTest.java` and `CookeryFan.java`

## Problem 3

Consider Homework 9 - Problem 3's `ScrollColorPlay.java`.

What if the user comes up with a color combo they really like? They might like to save its red-green-blue values to a file.

Modify `ScrollColorPlay.java` into `ScrollColorSave.java`, such that:

- there is now a button that, when clicked, brings up an appropriate `JOptionPane` that asks for the name of a file to which it should try to write the current red, green, and blue scrollbar values,
- and when such a file name is entered, a `PrintWriter` instance is used to try to write the current red, green, and blue scrollbar values to a file with the entered name, including at least the following:

- a descriptive heading
- the red scrollbar's current value with appropriate label-text on its own line
- the green scrollbar's current value with appropriate label-text on its own line
- the blue scrollbar's current value with appropriate label-text on its own line
- (what if a user happens to enter the same file name more than once? This can cheerfully overwrite the previous contents.)
- OPTIONAL variation: write this so that it appends to the given file rather than overwrites any previous contents
  - see the after-class aside on how to do this in the Week 14 Lecture projected notes if you are interested in this optional variation

Submit your resulting `ScrollColorSave.java`.

## Problem 4

Consider a text file of "inspirational" phrases, one phrase per line.

Consider a little application that seeks to inspire someone by randomly displaying a different one of these phrases every 5 seconds.

And, consider such an application that also allows the user to ADD phrases to the possible phrases as it is running.

Create a Swing application `InspireMe.java` that has the following requirements:

- It uses a `JOptionPane` to ask the user to enter the name of a file of phrases to be used.
- It uses a `Scanner` to attempt to read these phrases from the entered file and into an `ArrayList` of `String` instances.
  - Note: opening a file, even with `Scanner`, and reading it can throw more possible exceptions than one might expect!
  - IF an exception is thrown, your program should display a `JOptionPane` with a complaint and then exit.
  - You are permitted, if you'd like, to just catch `Exception` and do the above.

Or, if you'd like to look up the specific exceptions that might be thrown by the different methods -- which can be found in the Java 8 API, since each method's documentation specifies the types of exception it might throw -- that would be fine, too, if you'd prefer more-specific complaints in your `JOptionPane` message dialogs.
- It uses a separate thread to somehow PAINT or DISPLAY (your choice!) a randomly-chosen phrase from your `ArrayList` of phrases to your application's center panel approximately every 5 seconds.
  - the phrase's color, font, size, and location are all up to you; and it can be painted on the center panel or displayed in an uneditable `JTextField` or a `JLabel` in the center panel
  - OPTIONAL variation: use a `Timer` instead of a separate thread to modify the phrase displayed. See pp. 302-305 of the "Core Java" course text if you are interested in this variation.

- Include a subpanel in the south/page\_end with an "Enter new phrase:" label, a textfield where a new phrase can be entered, and an "Add" button.
- When the "Add" button is clicked, whatever is in the new-phrase textfield should be added to the `ArrayList` of phrases.
  - This should be done in such a way that, once you have added a new phrase, it has a chance to occasionally be selected to be painted to the panel.
- Also, somewhere in your layout, add a "Save" button that, when clicked, uses `JOptionPane` to ask the user to enter the name of a file where the current `ArrayList` of phrases should be saved, and it then attempts to write the current contents of the phrase collection to a file with that name (happily overwriting any current contents if a file with that name already exists).

Submit your resulting `InspireMe.java`.