COSC 1P03 Assignment 2

"Gutenberg never had to hand in an assignment before 4:00"

***Due: Feb. 27, 2017 @ 4:00 pm** (late date Mar. 2 @ 4:00 pm)*

In preparation for this assignment, create a folder called `Assign_2` for the DrJava project for the assignment. The objective of this assignment is to apply sequentially-linked structures in the development of a problem solution.

Print Spooler

Whenever you click to print a document, you probably briefly see an icon appear in the corner of the screen. That's your print spooler accepting the print job, so that it may be sent to the printer when it's available. If you quickly send multiple documents, the spooler will hold the additional jobs until they can be printed. In environments with shared printers, the spooler will typically be on a separate server, with all users sending jobs to it.

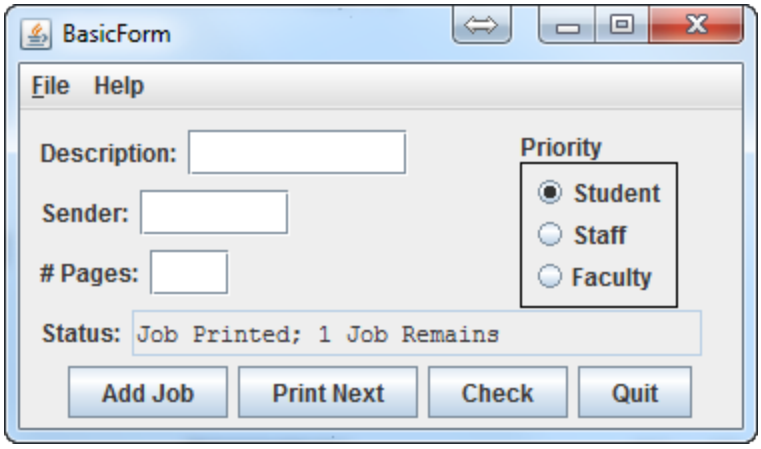
Firstly, this means that some users will occasionally need to wait for others to finish. Ostensibly, a *first-come, first-served* approach is used, but there can be another dynamic as well. Consider, for example, an academic environment.

- Faculty members (professors, lecturers, etc.) have more flexible due dates, so they can afford to wait the longest
- Staff members, on the other hand, need to produce work to support faculty, and so can "jump the queue". That is, a new staff print job will be printed before a faculty job that's been waiting
- Students have the strictest due dates, and so need to be able to submit documents very quickly. As such, student jobs are prioritized over staff (and thus also over faculty)
- For all of these, within the *same* level of priority, the *first-come, first-served* behaviour remains

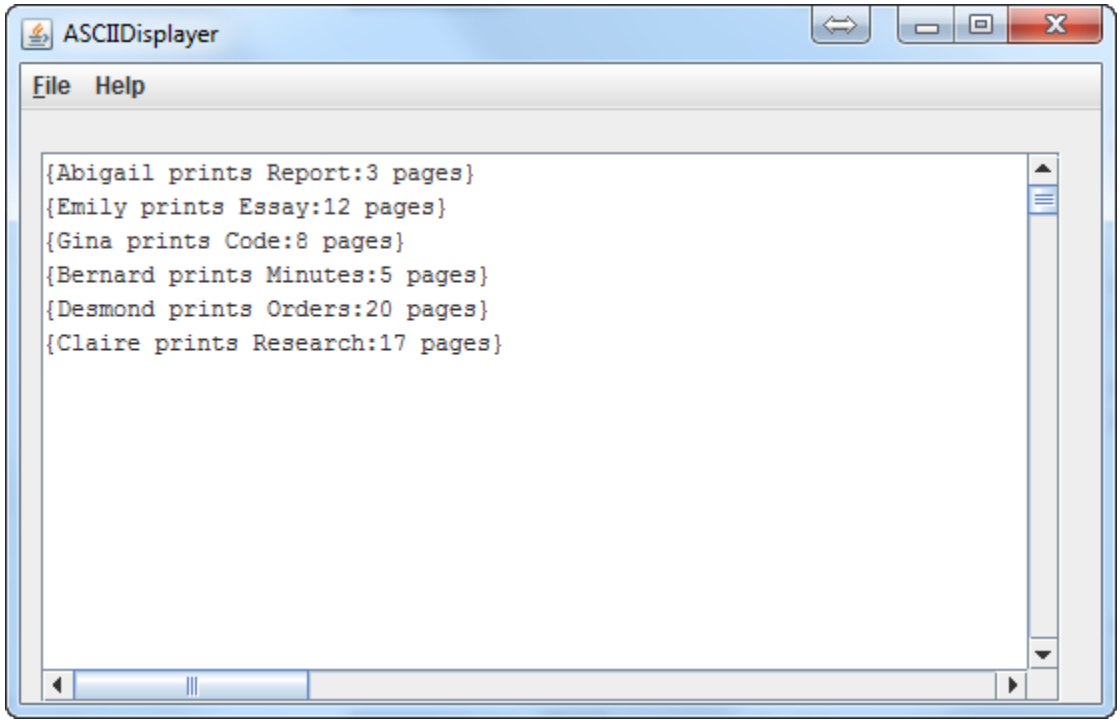
Of course, since one never knows how many print jobs will be backlogged, the spooler should be flexible, and grow *dynamically* with the number of jobs being stored.

Requirements

- You need at least three classes:
  - One for a node
  - One for a print job
  - One for the print spooler/main class
- Each job contains four important pieces of information:
  - A description of the job
  - The name of the sender
  - The number of pages in the job
  - A way of identifying the priority of the job (i.e. Student, Staff, or Faculty)
- You must create a `BasicForm`-driven application that keeps print jobs in a linked structure. The form for entering print job information should look similar to the following:



- The text fields on your form should be cleared after each added job
- You need a text field that is solely for feedback (`Status`), to confirm the most recent action
  - When a job is added, it should give confirmation
  - Whenever a job is printed, it give confirmation and indicate the number of jobs left to print
  - When the spooler is checked it should indicate the number of pages in all remaining jobs
- You also should use an `ASCIIDisplayer` to log the activity indicating, whenever a job is printed, the job information such as below:



Hints

- Since each job has several data fields, you should have a method to make reporting (to the `ASCIIDisplayer`) on a printed job easier
- Even though we're using words to describe the priority levels, that's only on the user-side; it might be easier to use numbers internally

- The `Status` text field is **never** used by the user for data entry, so editing shouldn't be possible
- One way of organizing jobs/nodes is to assume that printing will always print the job at the front of the list. If priority weren't an issue, that would mean each job should get added to the end of the list. Since priority matters, you'll be doing a *sorted insertion* that effectively adds to the end of the portion of the list consisting of jobs of the same priority
- You might find an incremental approach easiest. First start by adding to the end, and removing from the front
- Clicking `Print Next` when the spooler is empty (i.e. no jobs in the list) is not an error so a program crash isn't acceptable. You'll need to handle that special case (and provide appropriate feedback in `Status`)
- Take note that there are two different means of tallying how much is left (the total page count across all jobs when the `Check` button is pressed, and the number of jobs remaining after each print). You can either maintain running tallies, adjusting whenever a job is added/removed, or you can simply traverse the list to count whenever such a number is needed

Submission:

Details regarding preparation and submission of assignments in COSC 1P03 are found on the COSC 1P03 Sakai Site as `Assignment Guidelines` under `Course Documents`. This document includes a discussion of assignment preparation, programming standards, evaluation criteria and academic conduct (including styles for citation) in addition to the detailed assignment submission process copied below.

To prepare and submit the assignment electronically, follow the procedure below:

1. Ensure your folder (say `Assign_2`) for the assignment accessible on your computer.
2. Using DrJava, print (to `CutePDF Writer`) each of the `.java` files of your assignment using the name *`ClassName`*`.pdf` where *`ClassName`* is the class name (i.e. same name as the `.java` file) and save the `.pdf` file at the **top level** of the project folder (i.e. directly within `Assign_2`).
3. Run the program using the script of actions (next page). When you reach the end of the script (i.e. 1 job pending) but before you press `Quit` or `Close`, select `File/Print Image of Window`... on the `BasicForm` and print to `CutePDF Writer` as `form.pdf` at the **top level** of the project folder (i.e. directly within `Assign_1`). Do the same for the `ASCIIDisplayer` as `display.pdf`. The result should be similar to the above. `Quit` the application and `Close` the display.
4. The submission folder (`Assign_2`) should now include the `.java`, `.class` (created by DrJava) and `.pdf` files for the class you wrote. It should also include the `.pdf` files for the form and display as above.
5. Create a `.zip` file of your submission by right-clicking on the top level folder (i.e. `Assign_2`) and selecting `Send to/Compressed (zipped) folder`. A zipped version of the folder will be created. Use the default name (`Assign_2.zip`).

6. Log on to Sakai and select the COSC 1P03 site.
7. On the `Assignments` page select `Assignment 2`. Attach your `.zip` file (e.g. `Assign_2.zip`) to the assignment submission (use the `Add Attachments` button and select `Browse`). Navigate to where you stored your assignment and select the `.zip` file (e.g. `Assign_2.zip`). The file will be added to your submission. Be sure to check the `Honor Pledge` checkbox. Press `Submit` to submit the assignment. You should receive a confirmation email.
8. Assignments incorrectly submitted will lose marks. Assignments without the required files may not be marked.

DrJava

The `.zip` folder you submit should contain the project folder including all files relevant to the project—the `.drjava`, `.java` and `.class` files for the assignment and `.pdf` files for program listings and output.

Other Platforms

If you are using an IDE other than DrJava to prepare your assignment, you must include the `.java` source files and the `.pdf` files described above as well as a file (likely `.class` or `.jar`) that will execute on the lab machines.

Test Script

Use the following sequence to demonstrate your program:

- Add Abigail (Student), 3 page Report
- Add Bernard (Staff), 5 page Minutes
- Add Claire (Faculty), 17 page Research
- Add Desmond (Staff), 20 page Orders
- Add Emily (Student), 12 page Essay

(To verify it's working so far, check the current page count; it should be 57)

- Print a job
- Print another job

(It should show 3 jobs remaining; if you check, there should be 42 pages remaining)

- Add Fahad (Faculty), 2 page Paperwork
- Add Gina (Student), 8 page Code

(Verify that you have 52 pages waiting to print)

- Print 4 jobs

(There should be 1 job pending)

`Check` the number of pages remaining, and include in your submission the printed (to `.pdf`) copy of the form and display as described above.