# COSC 1P03 Assignment 2

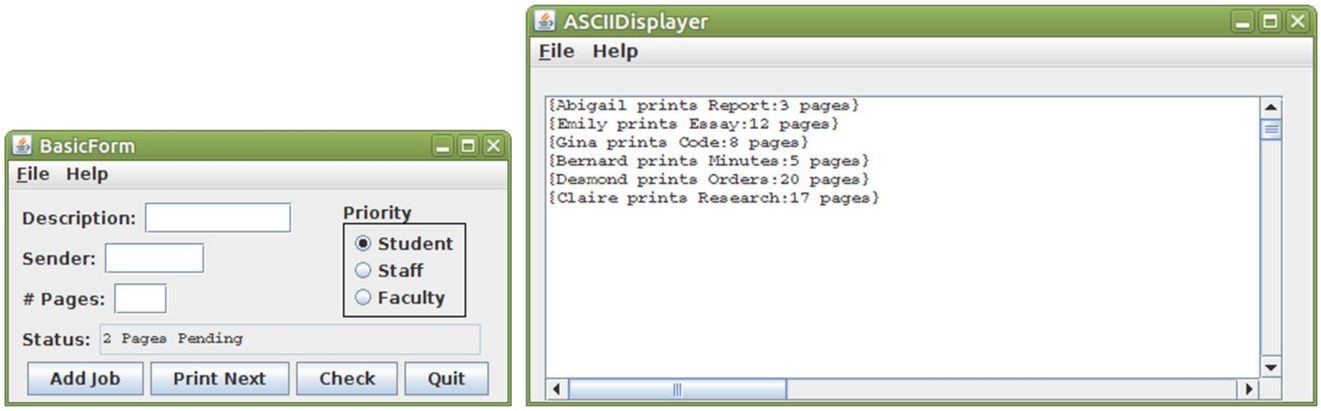
## "Gutenberg never needed to hand in an assignment before 4"

***Due: Feb. 24, 2020 @ 4:00 pm (late date Feb. 27 @ 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 use a dynamically sized structure to accept, hold, and release print jobs, with consideration to both sequence of addition and relative priorities of jobs.

## 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-serve* 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 most strict 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-serve* 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 must create a BasicForm-driven application that keeps print jobs in a linked structure
- You need at least three classes:
  - One for your *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)
- The text fields on your form should reset after each added job
- You need a text field that is solely for feedback, to confirm the most recent action
    - Whenever a job is printed, use this field to display the number of pages remaining
- One of your buttons should use the feedback to show the *total* number of pages in the spooler
- Also include an ASCIIDisplayer to fully log all jobs *as they are printed*

## Hints

- Since each job has several data fields, you should have a method to make reporting 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 feedback text field is *never* used by the user for data entry, so editing shouldn't be possible
- When adding instance variables, don't forget good programming practices, and visibility modifiers
- One way of organizing jobs/nodes is to assume that printing will always draw from the front of the list. If priority weren't an issue, that would mean each job would 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 to *Print* even when the spooler is empty is not an error; a program crash isn't acceptable, so you'll need to handle that special case (by simply providing appropriate feedback)
- 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 is accessible on your computer and contains the Dr. Java project and all associated `java` and `class` files for your assignment.

2. 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`). It is **important** that you only submit a `.zip` file, not `.rar` or `.tar` or any other type of compression. If you use a type of compression other than `.zip` your assignment may not be marked.
3. Log on to Sakai and select the COSC 1P03 site.
4. 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.
5. 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. If your project requires any special instructions to run, these instructions must be included in a read me file.

## Other Platforms

Students must create their project using an IDE that is available on the Brock Computer Science lab computers. Currently, these are NetBeans, IntelliJ and Dr. Java. Markers must be able to open, compile and run your project on the lab computers. Assignments completed using some other IDE may not be marked. It is the student's responsibility to ensure their project will load, compile and run on a lab computer.

## Test Script

Use the following sequence to test 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.

(At this point your form and displayer should match the screenshots on the first page.)