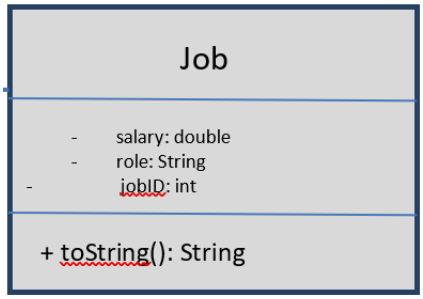
The purpose of this lab is to some file processing .



Part 1 – set up the Job class– 0.3

Set up a **Job** class as shown, noting the following:

- **Job** class constructor: set up all attributes with incoming values: salary, role , jobID.
- **Private** Attributes (generate getter & setter methods if your IDE does this automatically)
  Note: Add validation so that salary must be > 0 and less than 100000.
- Add a toString() method to the Job class to format a string of its attributes: Note – you'll need to convert numerics to Strings as discussed in class.

Test your code by instantiating sample Job objects and "system.out.println"ing the objects from a "main" method in another "Control" class (or whatever you decide to call it).

Part 2 – Working with files – reading - if complete 0.6

The **Job class constructor** now needs to validate that the "role" value being sent in to the constructor as follows: the role is valid if it exists in the text file, "roles.txt" – this file is in Brightspace. To avoid hard coding file paths, put the "roles.txt" file goes into your project directory in Eclipse (not src or package).

To implement this, it is best to put the file processing functionality into a separate class, as reading/ manipulating files is not core to the job class. The job class will use this file processing class.

Create a separate FileProcessor class that has methods to

(1) A constructor to set up a FileProcessor object (Suggest have one attribute : String filename);
(2) Method to connect to the file (creates a File object);
(3) Method to read the file and return a string array

Then *use this FileProcessor class from your setRole(String role) method in the Job* class to validate the "role" value being sent into the constructor exists in the roles file (i.e. exists in the String array of values returned by the File Processor class.

Hint: equals() method of the String class is the way to compare two string values.

Part 3 – Using a static variable 0.8

Job Id will now be assigned as sequential values - It will assigned to job objects as 100, 200, 300, 400.. Implement a static attribute that stores the value of jobID (i.e. acts as a counter) and assigns it in the setRole() method.
Note that one of the Employee attributes is "job" – i.e. is an object of type "Job".

Test your code by instantiating sample Job objects, and calling System.out.println on the objects to make sure the jobID is being allocated correctly.

Part 4 – Writing to a file – full marks

The Job class now needs to write every new JobID into a file. The toString() method of the Job class should write the JobIDs to a **file**– "jobs.txt" - as well as returning a String.
To do this, edit your **FileProcessor** class so that it can be used by the **Job** class to write text to a file. Suggested methods (although you can if you prefer also combine these into a single writeLine method that does all the open, write, close – but better to separate out):

- A getFileWriter that creates a printwriter object;

- A writeLine method that writes a string to a file
- A closeWriter method that closes the printWriter

To test your code, create a job object from the control class, and System.out.println the object to run the toString() method, where your file writing functionality is used.

You'll be able to see the "jobs.txt" file in the project directory to see that the write has worked.