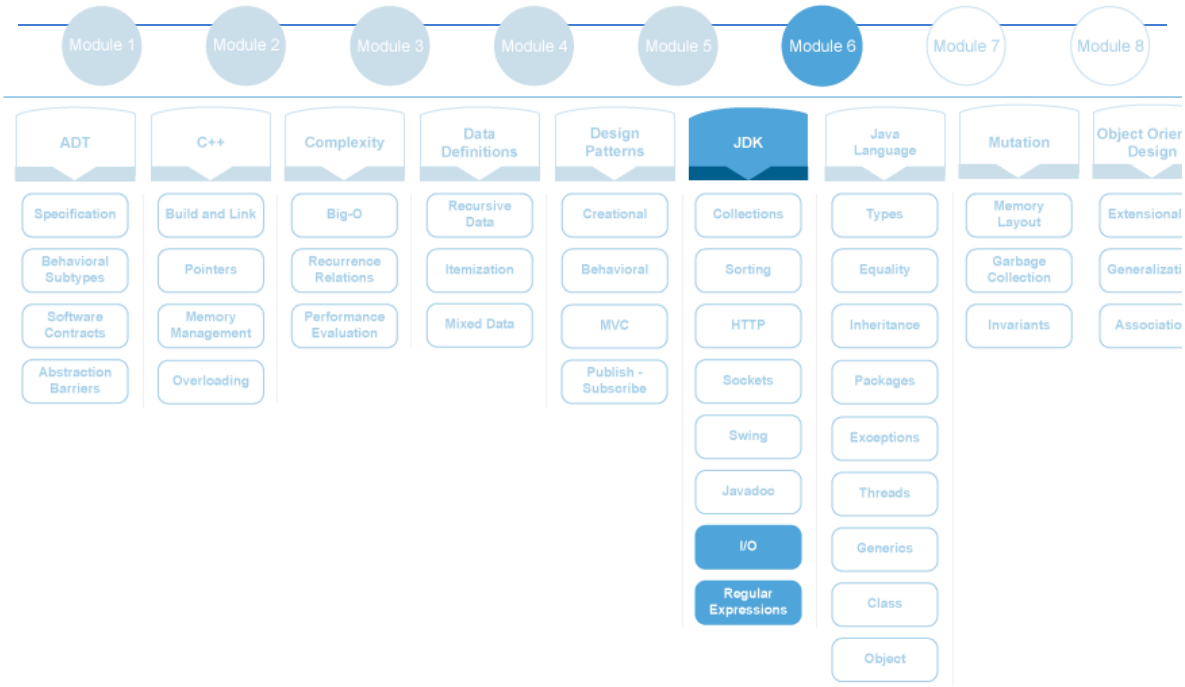
CS 5004: Object Oriented Design and Analysis

Northeastern University
College of Computer and Information Science

- Overview
- Course Map
- Objectives

In this module, we start our conversation about regular expressions, and we show how to use regular expressions in Java.

We then start start our conversation about functional programming Java, and introduce streams, functional objects, higher order functions, as well as lambda functions. We introduce different types of higher order functions, and introduce syntax for lambda functions. We consider multiple simple examples of functional programming in Java 8.



1. Given a search specification, write down it's corresponding regular expression.
2. Write a Java program that accepts input parameters on the command line.
3. Write a Java program that uses Java's regular expression to search for patterns in a String.
4. Write a Java program that reads data from a file.
5. Write a Java program that writes data to a file.
6. Write a simple lambda function in Java.
7. Write a simple Java program that takes some data collection, and converts it into a stream.
8. Write a Java program that takes a stream, and converts it into some data collection.
9. Write a Java program that takes a stream, manipulates it, and releases the results of that manipulation.

- Reading
- Assignment 9

Java Tutorial: Regular Expressions
Java Tutorial: Nested Classes
Flying Bytes: An Introduction to Functional Programming in Java 8 - Motivation
Flying Bytes: An Introduction to Functional Programming in Java 8 - Functions as Objects
Flying Bytes: An Introduction to Functional Programming in Java 8 - Optionals
Flying Bytes: An Introduction to Functional Programming in Java 8 - Streams
Flying Bytes: An Introduction to Functional Programming in Java 8 - Splitters

DUE DATE: Friday, April 6th at 11:59pm

Submission Criteria

Repository Contents

Your repositories must contain only the following:

1. pom.xml file and
2. src folder with appropriate sub-folders with your code and test code only.

Code Criteria

1. Your solution should be in its own package. The package name should follow naming convention: `edu.neu.ccs.cs5004.assignmentN.problemM` where you replace `N` with the assignment number and `M` with the problem number, e.g., all your code for problem 1 for this assignment must be in a package named `edu.neu.ccs.cs5004.assignment9.problem1`.
2. Your project should successfully build using Maven, and it should generate all the default reports.
3. Your Javadoc generation should complete without errors or warnings.
4. Your Checkstyle report must not have any violations.
5. Your JaCoCo report must indicate 70% or more code coverage per package for "Branches" **and** "Instructions".
6. Your FindBugs report must not have violations (except for violations categorized as PERFORMANCE or SECURITY, you can ignore those).
7. Your PMD report should have no violations.

Theater Company Email Automation

You are asked to automate the process that an independent theater company uses to communicate with its members. Every year, the theater company holds special, members-only, showing of their most popular play. The company sends an email informing each member of the play and the dates. Then it also sends complimentary tickets using normal mail. Over the last several years, the company has grown significantly, and their manual process is becoming time consuming. They are asking you to help them automate the process.

The theater company holds all the information about their members in a CSV file. CSV file is organized as a plain text file, containing information such that each piece of data is enclosed in double quotes, and separated by a comma. The first line of the file contains the headers for each column.

```
"first_name" , "last_name" , "company_name" , "email"
"James"      , "Butt"      , "Benton"       , "504-845-1427"
"Josephine"  , "R, Darakjy", "Chanay"       , "810-374-9840"
"Art"        , "Venere"    , "Chemel"       , "856-264-4130"
```

For example, the preceding listing has 4 columns named `first_name`, `last_name`, `company_name` and `email`. The second line has the information for member `James Butt`. Note that even though information is enclosed in

double quotes and separated by comma, within the double quotes, there may exist column entries that contain comma, e.g., `"R, Darakjy"` is one valid piece of information, and not two.

Here is a sample that you can use with some of the theater company's members information theater-company-members.csv. The CSV file contains first and last name, company name, address, city, county, state, zip, phone1 and phone2, email address and web page URL.

Given this CSV file, the theater company would like you to create a program that they can run on the command line. The program would take this file as input and generate files that will contain the email messages and letters to send to their members. Email messages and letters will be generated from **templates**.

The templates are stored as text files with special placeholders in the text that refer to the CSV file's header names. Here are two example templates, one for email and one for letters. Placeholders are placed between `[[` and `]]`

```
To:[[email]]
Subject:Information on this years members only show!

Dear [[first_name]] [[last_name]],

   This year's members only theater show will showcase "A Streetcar
   Named Desire" directed by John Jarmush and Susan Mae at our New
   York location between March 1st and April 10th.  Your complementary
   tickets for the show are on their way through mail and should
   reach you within the next couple of days.

   Sincerely,
```

So given the above email template and the following line from the CSV file:

```
"first_name","last_name","company_name","address","city","county","state","zip","phone1","phone2","email"
"Art","Venere","Chemel, James L Cpa","8 W Cerritos Ave #54","Bridgeport","Gloucester",
"NJ","08014","856-636-8749","856-264-4130","art@venere.org","http://www.chemeljameslcpa.com"
```

the email message that is generated looks like:

```
To:art@venere.org
Subject:Information on this years members only show!

Dear Art Venere,

   This year's members only theater show will showcase "A Streetcar
   Named Desire" directed by John Jarmush and Susan Mae at our New
   York location between March 1st and April 10th.  Your complementary
   tickets for the show are on their way through mail and should
   reach you within the next couple of days.

   Sincerely,
```

Similarly, we also have a template for the company's letter:

```
[[company_name]].
[[first_name]] [[last_name]]
[[address]], [[city]],
[[county]], [[state]], [[zip]]
([[email]])
```

```
Dear [[first_name]] [[last_name]],

    Please find enclosed your complementary tickets to "A Streetcar
    Named Desire" directed by John Jarmush and Susan Mae. We look
    forward to seeing you at one of our showings at our New York
    theater between March 1st and April 10th.

Sincerely,
```

For the same CSV lines we used before, this template generates the following text file:

```
Chemel, James L Cpa,
Art Venere
8 W Cerritos Ave #54, Bridgeport,
Gloucester, NJ, 08014.
(art@venere.org)

Dear Art Venere,

    Please find enclosed your complementary tickets to "A Streetcar
    Named Desire" directed by John Jarmush and Susan Mae. We look
    forward to seeing you at one of our showings at our New York
    theater between March 1st and April 10th.

Sincerely,
```

Note: in this assignment, you are provided with **one** example CSV file and **two** examples of templates. Your code should work for **any** CSV file and **any** template that uses the CSV file's header values as placeholders.

The theater company would like to write more templates and your program should accommodate for new templates. Templates use the column names from the CSV file as the names to replace inside placeholders, like the examples provided here.

Your program needs to accept certain arguments at the command line.

```
--email                  only generate email messages
--email-template <file>  accepts a filename that holds the email template

--letter                 only generate letters
--letter-template <file> accepts a filename that holds the email template


--output-dir <path>      accepts the name of a folder, all output is placed in this folder
--csv-file <path>        accepts the name of the csv file to process
```

Some options take arguments, for example `--email-template` takes one argument, and it is the name of a file, `--output-dir` takes one argument, and it is the name of a folder. Other options take no arguments, and indicate an action, i.e., `--email` indicates that we should generate emails on this execution of the program.

The command line option `--output-dir` and `csv-file` are required. Your program should be able to generate one of the two options (emails or letters) per invocation. If `--email` is given, then `--email-template` must also be provided, if `--letter` is given then `--letter-template` must also be given. Calling your program and passing any other combination of options should generate an error, e.g.
`--email --letter-template letter-template.txt --output-dir letters/` is illegal.

When a user provides an illegal combination of inputs, the program should exit with a helpful error message, and a short explanation of how to use the program along with examples. For example passing

```
--email --letter-template letter-template.txt --output-dir letters

Error: --email provided but no --email-template was given.

Usage:

       --email                       only generate email messages
       --email-template <file>  accepts a filename that holds the email template.
         Required if --email is used

       --letter                       only generate letters
       --letter-template <file> accepts a filename that holds the email template.
         Required if --letter is used

       --output-dir <path> accepts the name of a folder, all output is placed in this folder

       --csv-file <path> accepts the name of the csv file to process

Examples:

      --email --email-template email-template.txt --output-dir emails --csv-file customer.csv
      --letter --letter-template letter-template.txt --output-dir letters --csv-file customer.csv
```

Finally, please note the order of the command line options does not matter, i.e. the following examples are valid:

```
--email --email-template email-template.txt --output-dir emails --csv-file customer.csv
--csv-file customer.csv --email-template email-template.txt --output-dir emails --email
--output-dir emails --email --csv-file customer.csv --email-template email-template.txt
```

Please design and implement the email and the letter generator program for the theater group. Use theater-company-members.csv, email-template.txt and letter-template.txt to help you develop and test your code. You should also develop your own templates. Also make sure that your program works correctly regardless of how your operating system represents paths and files.