Reading Files using Java

In this lecture we will discuss:

- Scanner review
- Input/Output Streams in Java
- FileInputStream and the File class
- Reading in from text based files.

Your future in CS

I used to include this on my slides, but since these slides have changed - going to just leave it up here for every notebook. I get a lot of questions about more programming courses, the concentrations, and minors in computer science. Here is a brief reminder.

CS 165 – Next Course In Sequence, also consider CS 220 (math and stats especially)

- CO Jobs Report 2021 – 77% of *all* new jobs in Colorado require programming
- 60% of all STEM jobs requires *advanced* (200-300 level)
- 31% of all Bachelor of Arts degree titled jobs also required coding skills
- 2016 Report found on average jobs that require coding skills paid $22,000 more

- Concentrations in CS:
    - Computer science has a number of concentrations.
        - General concentration is the most flexible, and even allows students to double major or minor pretty easily.
        - Software Engineering
        - Computing Systems
        - Human Centered Computing
        - Networks and Security
        - Artificial Intelligence
        - Computer Science Education.
    - Minors:
        - Minor in Computer Science - choose your own adventure minor
        - Minor in Machine Learning - popular with stats/math, and engineering
        - Minor in Bioinformatics - Biology + Computer Science

Reading Review:

Given the following code, answer the questions.

In [12]:
```
String line = "Let me be that I am and seek not to alter me.\n-Much Ado About Nothing"
```

```
Scanner scanner = new Scanner(line);
int tokenCounter = 0;
while(scanner.hasNext()) {
    tokenCounter++;
    System.out.println(scanner.next());
}

System.out.println(tokenCounter); // question 1
```

```
Let
me
be
that
I
am
and
seek
not
to
alter
me.
-Much
Ado
About
Nothing
16
```

Question 1:

What is printed on the question 1 location?

Question 2:

What is the default token for scanner?

A. Single Space

B. Any amount of whitespace (Space, tab, line return, etc)

C. Comma

D. Trick question, there is no "default"

In [13]:
```
line = "Let me be that I am and seek not to alter me.\n-Much Ado About Nothing";
scanner = new Scanner(line);
int lineCounter = 0;
while(scanner.hasNext()) {
    lineCounter++;
    scanner.nextLine();
}
System.out.println(lineCounter); // question 3
```

```
2
```

Question 3

How many lines are there?

```

```

In [10]:
```
scanner = new Scanner(line);
scanner.useDelimiter("-");
int otherCounter = 0;
while(scanner.hasNext()) {
    otherCounter++;
    System.out.println(scanner.next());
}
```

```
Let me be that I am and seek not to alter me.

Much Ado About Nothing
```

Question 4

How many tokens are there?

Question 5

Including the \n added with .println(), how many new line characters are printed?

Scanner

- A scanner is a tool designed to for reading Streams or Strings.
- Default way to split the stream into "tokens" is any amount of whitespace
  - A token is a single element in the sequence
  - `.next()` gives the next token in the sequence
- You can change the split by using `.useDelimiter("-")` ;

So let's go back to our Coordinates from Friday.

In [20]:
```
String location = "Fort Collins,40°35'6.9288\"N,105°5'3.9084\"W";
Scanner locScan = new Scanner(location);
locScan.useDelimiter(",");
String city = locScan.next();
String lat = locScan.next();
String lon = locScan.next();

System.out.println(city);
System.out.println(lat);
System.out.println(lon);
```

```
Fort Collins
40°35'6.9288"N
105°5'3.9084"W
```

Scanner Methods

There are a number of useful methods in Scanner

- next()
- hasNext()
- nextLine()

- hasNextLine()
- hasNextInt() - also hasNextDouble/Long/etc
  - This says the token can be parsed into the primitive of the matching type
- nextInt() - also nextDouble(), etc
  - Takes the next token, and attempts to convert it to the specified type
  - Potential flaw: what if the next type is not an int/double/etc?
  - Throws a `NumberFormatException`

In practice that means we do the following:

In [7]:
```
Scanner scanner = new Scanner(System.in);
System.out.println("Enter a number: ");
while(! scanner.hasNextInt()) {
    String input = scanner.next();
    System.out.println(input + " is not a number, please try again: ");
}
int val = scanner.nextInt();
System.out.printf("You entered %d, with doubled is: %d", val, val*2);
System.out.println();
```

```
Enter a number:
The is not a number, please try again:
answer is not a number, please try again:
is is not a number, please try again:
You entered 42, with doubled is: 84
```

Input Streams

- `System.in` is an **Input Stream**
  - A stream of data coming *in* from the client
- `FileInputStream` or `File`
  - Are also input streams
  - But they read from a file
- Other Streams?
  - Not covered in this class, but networked data is another common one.

Reading files is extremely common, and the format / common syntax is as follows:

```
try {
    Scanner fileIn = new Scanner(new File("Name of file as a String"));
    // now the file is in a scanner, and looping matters!
    while(fileIn.hasNextLine()) { // could also use hasNext()
        System.out.println(fileIn.nextLine());
    }
}catch(IOException ex) {
    // there was an error finding the file or reading the file!
    // so how do you hand that? For now, we just say that and end the
program
    System.err.print("Error reading file!");
    ex.printStackTrace();
```

```
    System.exit(1);
}
```

Let's try it!

In [13]:
```
try {
    Scanner fileLocations = new Scanner(new File("data/poem2.txt"));
    while(fileLocations.hasNextLine()) {
        String line = fileLocations.nextLine(); // now I can just treat it as a string
        System.out.println(line);
    }
}catch(IOException ex) {
    System.out.println("Error reading file!");
    ex.printStackTrace();
    System.exit(1);
}
```

```
Because I could not stop for Death
by Emily Dickinson

Because I could not stop for Death,
He kindly stopped for me;
The carriage held but just ourselves
And Immortality.
```

In class Activity

Write a program that simply loads a file **into a String** - and then prints out

1. the length of that string
2. the contents of the file

In [23]:
```
String contents = "";
try {
    Scanner scn = new Scanner(new File("data/messages.txt"));
    while(scn.hasNextLine()) {
        contents += scn.nextLine() + "\n";
    }
}catch(IOException ex) {
    System.err.print("Error reading file");
}
System.out.println("Total Characters in file: " + contents.length());
System.out.println(contents);
```

```
Total Characters in file: 216
TO: Alice
FROM: Hatter
SUBJECT: Red Queen
BODY: To keep ahead, run from the red queen.
TO: Hatter
FROM: Red
SUBJECT: New Hat
BODY: Hatter, I need a new hat, and if I don't get it, you won't need one. *Kisses* Q
ueen.
```

Combing Objects and Reading Files

Let's actually do something with the poem, and make it more useful.

In [20]:
```
public class Poem {
    private String author;
    private String title;
    private String body;

    public Poem(String title, String author, String body) {
        this.author = author;
        this.title = title;
        this.body = body;
    }
    public String getAuthor() {return author;}
    public String getTitle() { return title;}
    public String getBody() { return body;}

    public String toString() { return String.format("%s\nby %s", getTitle(), getAuthor

    public static Poem loadPoemFromFile(String filename) {
        String title = null;
        String author = null;
        String body = "";
        try {
            Scanner scanner = new Scanner(new File(filename));
            if(scanner.hasNextLine()) title = scanner.nextLine();
            if(scanner.hasNextLine()) author = scanner.nextLine();
            while(scanner.hasNextLine()) {
                String line = scanner.nextLine().trim();
                if(!(body == "" && line.equals(""))) { // skip any empty lines before
                   body += line + "\n"; // because the new lines are removed, and we c
                }
            }
        }catch(IOException ex) {
            System.out.println("Error reading poem");
            return null; // leave the method early
        }
        return new Poem(title, author, body);
    }
}

Poem emily = Poem.loadPoemFromFile("data/poem2.txt");
Poem maya = Poem.loadPoemFromFile("data/poem.txt");

System.out.println(emily);
System.out.println(maya);
```

```
Because I could not stop for Death
by Emily Dickinson
Still I Rise
by Maya Angelou
```

In Class Activity

You will see `Student.java` and `Roster.java` in the in class activity - or on the github for this week.

1. In Student.java

- complete the getEName() method
  - It will take the first part of an email (before the @ sign)
  - Returns the lowercase version

2. In Roster.java
   - You will want to complete the 'readRoster(String filename)' method, which reads in the data from a file
   - Parse the incoming lines to build Student.java ArrayList
   - Use your helper methods!

In [37]:
```
public class Student {
    private String name;
    private String email;

    public Student(String name, String email) {
        this.name = name;
        this.email = email;
    }
    public String getName() { return name;}
    public String getEmail() { return email;}
    public String getEName() {
        return email.substring(0, email.indexOf("@"));
    }

    public String toString() {
        return String.format("%s, EName:%s, Email:%s",
                             getName(), getEName(), getEmail());
    }

}
```

In [38]:
```
public class Roster {
    private final ArrayList<Student> students = new ArrayList<>();
    public final String name;

    public Roster(String classname, String filename) {
        this.name = classname;
        readRoster(filename);
    }

    public void printRoster() {
        for(int i = 0; i < students.size(); i++) { // why did I use for, instead of fo
            System.out.printf("%d: %s\n", i+1, students.get(i));
        }
    }

    private void readRoster(String filename) {
        try {
            Scanner scn  = new Scanner(new File(filename));
            while(scn.hasNextLine()) {
                Student student = parseLine(scn.nextLine());
                students.add(student);
            }
        }catch(IOException ex) {
            System.err.println("Error reading file: " + filename);
        }
    }
```

```
    private Student parseLine(String line) { // notice i built a helper method!
        String name = line.substring(0, line.indexOf(","));
        String email = line.substring(line.indexOf(",")+1);
        return new Student(name, email);
    }

}

Roster roster = new Roster("Mystery 101", "data/students.csv");
roster.printRoster();
```

```
1: Scoobert "Scooby" Doo, EName:scooby_snack, Email:scooby_snack@mysteryinc.com
2: Norville "Shaggy" Rogers, EName:shaggy, Email:shaggy@mysteryinc.com
3: Daphne Blake, EName:not_helpless, Email:not_helpless@mysteryinc.com
4: Velma Dinkley, EName:mysterylover, Email:mysterylover@mysteryinc.com
5: Fred Jones, EName:trapman, Email:trapman@mysteryinc.com
```

Overall

You really want to focus on **divide**-**conquer**-**glue**, and treat the lines **as Strings**

- Which means - read the line, and then parse the String

We will come back to file writing (OutputStream) and try/catch in a couple weeks.

Adding files greatly expands what we can do with programs!