Regular Expressions and ~~Testing From State Machines~~ BNF

Lecture Objectives:

1) Explain the concept of BNF (Backus Normal Form or Backus–Naur Form) representation for grammar definition.
2) Define the concepts of a start symbol, a non-terminal symbol, and terminal symbol within a grammar.
3) Explain the concept of a rule in BNF.
4) Compare and contrast terminal symbil coverage, production coverage, and derivation coverage.
5) Explain the practical limitation to using derivation coverage in testing.
6) Define the concept of a ground string.

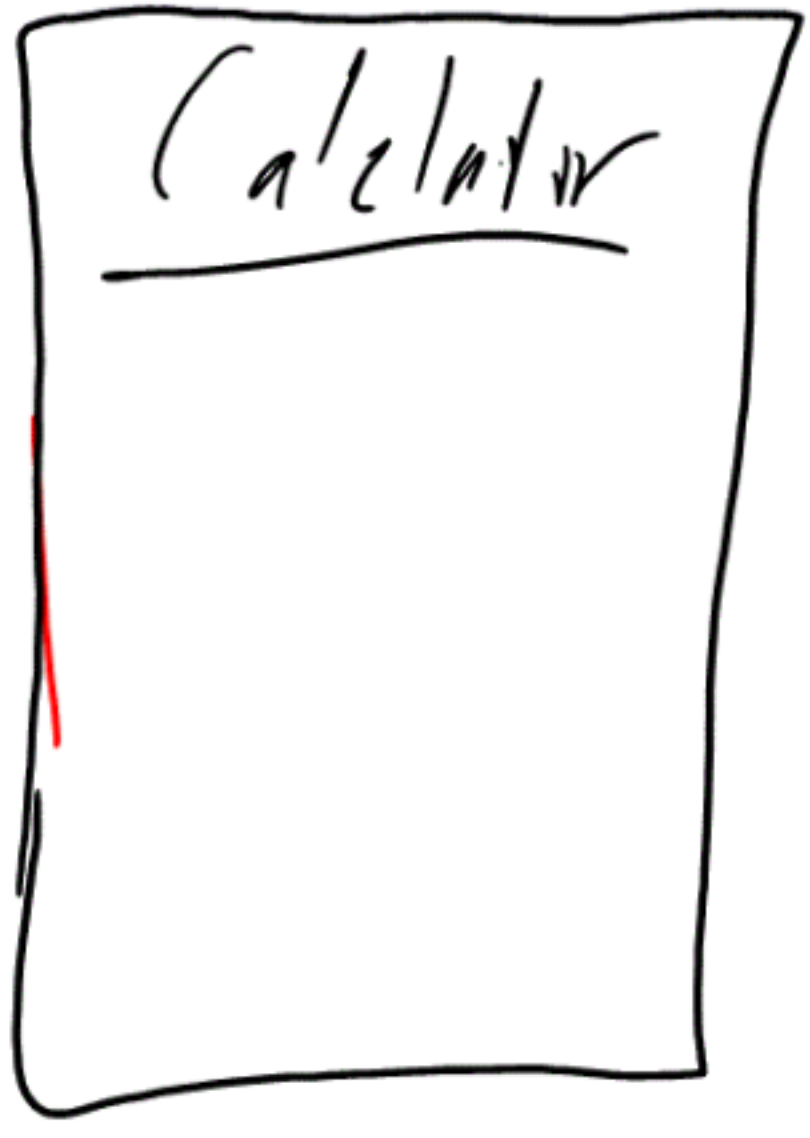
Design Example

- I want you to design and implement a command line calculator.
  - Allows the user to add, subtract, multiply, divide, and raise to a power.
- Talk with your neighbor.
  - How are you going to build it?

Design Example: Consensus on how to build it

2+2 6-8

4*4 8/16



nextLine()

...

if ...

add

else

subtract

...

MSOE

Lets break our design model.

$(2+2)^1 2$

$(2*2)\%(2^4) - 16*32$

Definitions

- Grammar
  - A grammar defines the allowable input formats for a program
- Regular Expression (regex)
  - A regular expression is a sequence of characters that forms a template used to search for strings [Words, phrases, or just about any sequence of characters.] within text. (http://linux.about.com/cs/linux101/g/regularexpressi.htm)

definition of acceptable inputs

Grep

String searching

Example Regular Expression

- (D n a | W n a | I n)*
  - D / W / I Commands to the program
  - N and a are arguments to the commands

entries → Type or digit

* closure operation – indicates zero or more occurrences

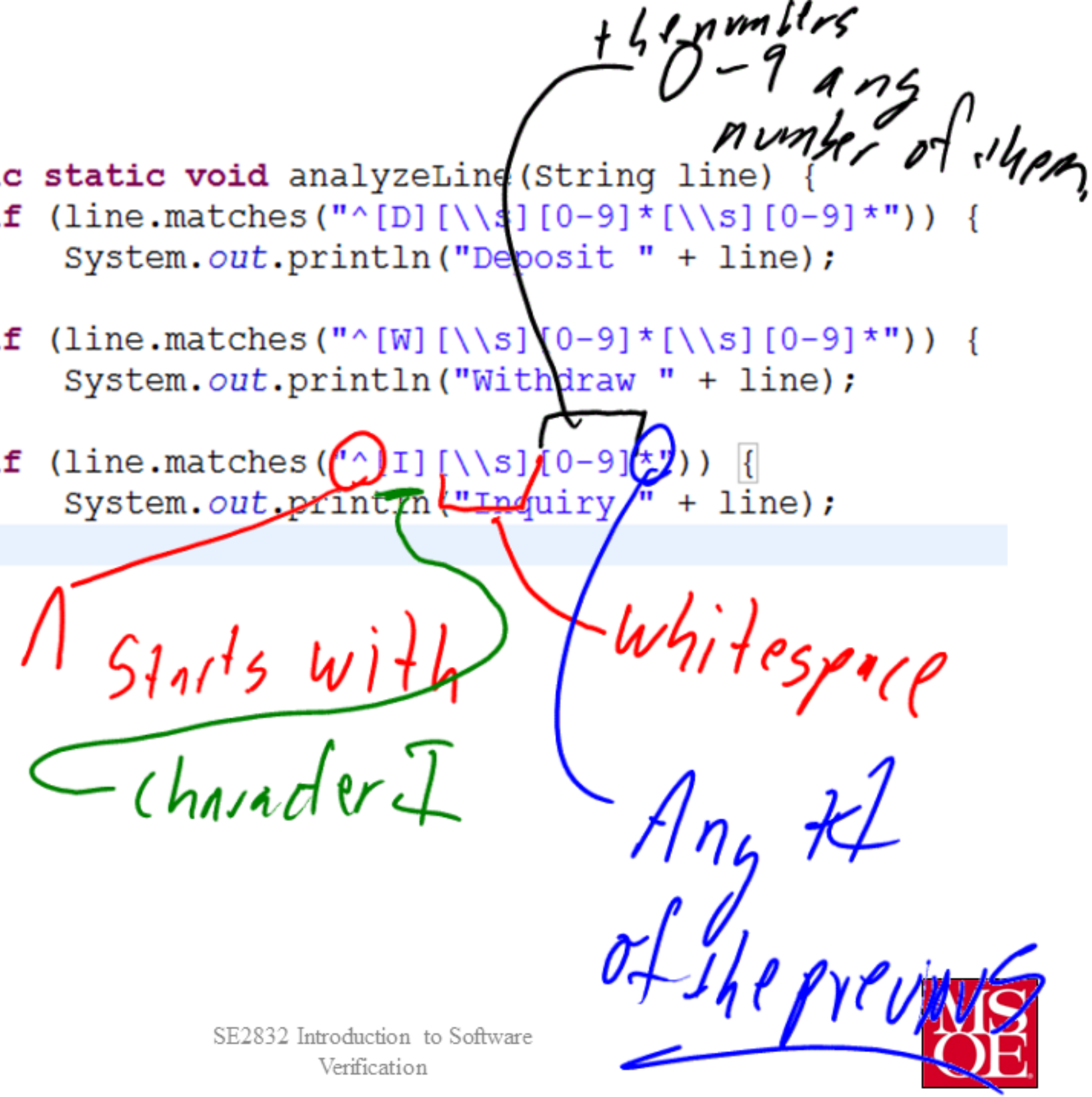
Vertical bar is choice operator (one or the other)

Example Code

```
public static void analyzeLine(String line) {
    if (line.matches("^[D][\\s][0-9]*[\\s][0-9]*")) {
        System.out.println("Deposit " + line);
    }
    if (line.matches("^[W][\\s][0-9]*[\\s][0-9]*")) {
        System.out.println("Withdraw " + line);
    }
    if (line.matches("^[I][\\s][0-9]*")) {
        System.out.println("Inquiry " + line);
    }
}
```

Example Code

```
public static void analyzeLine(String line) {
    if (line.matches("^[D][\\s][0-9]*[\\s][0-9]*")) {
        System.out.println("Deposit " + line);
    }
    if (line.matches("^[W][\\s][0-9]*[\\s][0-9]*")) {
        System.out.println("Withdraw " + line);
    }
    if (line.matches("^[I][\\s][0-9]*")) {
        System.out.println("Inquiry " + line);
    }
}
```



Context-free grammar (CFG)

- formal grammar in which every production rule is of the form

- $V \rightarrow w$

V ⇒ Non-terminal symbol

W ⇒ one or more terminals

'a' 'b' 'c' 'd' ....

name

Terminal symbols are the elementary symbols of the language defined by a formal grammar. *Nonterminal symbols* (or *syntactic variables*) are replaced by groups of terminal symbols according to the production rules.

Recognizers and Generators

- Recognizer
  - A program which decides whether or not a given string (or test case) is in the grammar

Java Compiler

- Generator – Makes Valid Strings
  - A tool which derives a set of terminals from a grammar which represents valid entries

EA

Coverage Related to Symbols

- Terminal Symbol Coverage
  - TR contains each terminal symbol in the grammar G
- Production Coverage
  - TR contains each production p in the grammar G
- Derivation Coverage
  - TR contains every possible string that can be generated from the grammar G

Our Example Again

- (D n a | W n a | I n)*
  - 14 terminal symbols

14 test cases

D 0 0.5

0 1 2 3 4 5 6 7 8 9 . D W I

D01
D02
D03
D04
D05
D06
D07
D08
D09
W01
I S

Getting back to our calculator (Int's)

expr = term | expr "+" term | expr "-" term

term = factor | term "*" factor | term "/" factor

10^10 10^10*10^10 10^10 / 10^10

factor = atom | atom "^" factor

10 10 ^ 10

atom = number | "(" expr ")“

10

number = digit $^{1\text{-}n}$ ⇒ 10

digit =‘0’ ‘1’ ‘2’ ‘3’ ‘4’ ‘5’ ‘6’ ‘7’ ‘8’ ‘9’

Let's develop some tests

2+2

2^2

2/2

2*2

2-2

2-(2+2)

2^(2-2)

2x2-2

2-2*2

2-2*2/2

(2-2)*2

2-2^2/2*2

2/(2-2)

2*(2-(2-2))