PDef
(**P**arenthesized **Def**initions)

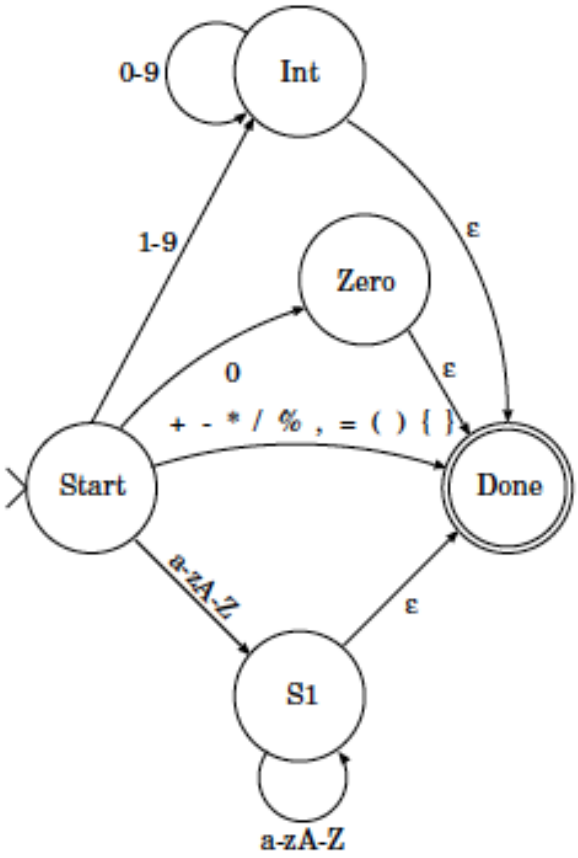
```
{ float a, a = 3, { int b, b = 4, a = b*a }, a = a+4.0 }
```

PDef: Parenthesized Definitions

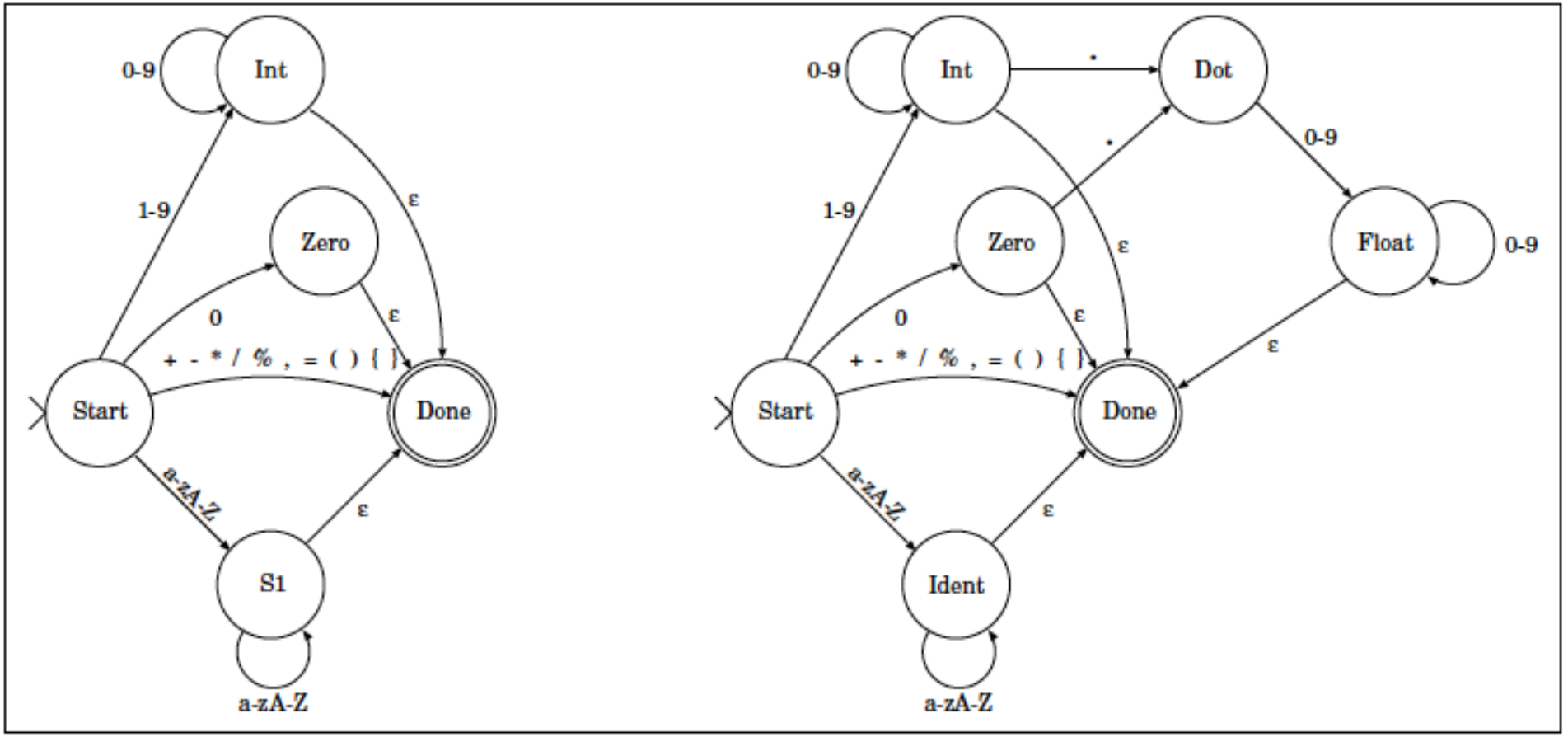
```
{ float a, a = 3, { int b, b = 4, a = b*a }, a = a+4.0 }
```

Token Class	Regular Expression	Termination Characters
addT	+	any character
subT	-	"
mulT	*	"
divT	/	"
modT	%	"
commaT	,	"
assignT	=	"
lpT	(	"
rpT	)	"
lcbT	{	"
rcbT	}	"
typeT	int | float	non-letter
intT	$0 \mid [1-9][0-9]^*$	non-digit
fltT	$(0 \mid [1-9][0-9]^*) \ . \ [0-9]^+$	non-digit
identT	$[a-zA-Z]^+$	non-letter

FSM for PDef



FSM for PDef


0-9
Int
1-9
Zero
ε
0
ε
+ - * / % , = ( ) { }
Start
Done
a-zA-Z
ε
S1
a-zA-Z
0-9
Int
.
Dot
.
0-9
1-9
Zero
ε
Float
0-9
0
ε
ε
+ - * / % , = ( ) { }
Start
Done
a-zA-Z
ε
Ident
a-zA-Z

Theory to Practice

- Need to represent the states, represent transitions between states, consume input, and restore input
- Create an enumerated type whose values represent the FSM states: Start, Int, Float, Zero, Done, Error, …
- Keep track of the current state and update based on the state transition

```
state = Start;
while (state != Done) {
   ch = input.getSymbol();
   switch (state) {
        case Start: // select next state based on current input symbol
        case S1:    // select next state based on current input symbol
        ..
        case Sn:    // select next state based on current input symbol
        case Done:  // should never hit this case!
   }
}
```

```
while (state != StateName.DONE_S) {
    char ch = getChar();
    switch (state) {
        case START_S:
             if (ch == ' ') {
                  state = StateName.START_S;
             }
             else if (ch == eofChar) {
                  type  = Token.TokenType.EOF_T;
                  state = StateName.DONE_S;
             }
             else if ( Character.isLetter(ch) ) {
                  name += ch;
                  state = StateName.IDENT_S;
             }
             else if ( Character.isDigit(ch) ) {
                  name += ch;
                  if (ch == '0') state = StateName.ZERO_S;
                  else state = StateName.INT_S;
             }
             else if (ch == '.') {
                  name += ch;
                  state = StateName.ERROR_S;
             }
             else {
                  name += ch;
                  type = char2Token( ch );
                  state = StateName.DONE_S;
             }
              break;
```

Project 1: Tokenizer for PDef

- Essentially, we are following along with the chapter 12 tutorial. I provide specific details / hints in the README
- Starter code is on GitHub
- Already a 'working' Java program (runs, but not correct output)
- Consists of:

```
├── .gitignore
├── PDef.java
├── README.md
├── debug
│   ├── Debug.java
│   └── TokenizerDebug.java
├── test
├── test2
└── tokenizer
    ├── Token.java
    └── Tokenizer.java
```

You are responsible for testing your code with various input (make new test files) and for submitting your working program to GitHub by the due date. **Programs that do not compile will not receive any credit.**

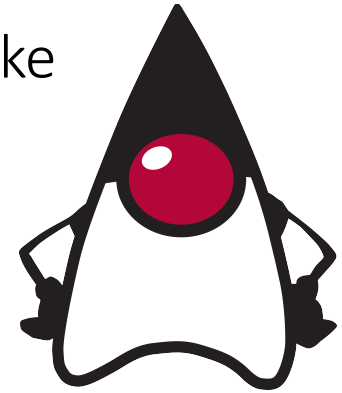

Java™

A brief history

- Developed by James Gosling (”Dr. Java”) at Sun Microsystems
- 1990s: First release
- 2000s: Sun Microsystems was acquired by Oracle

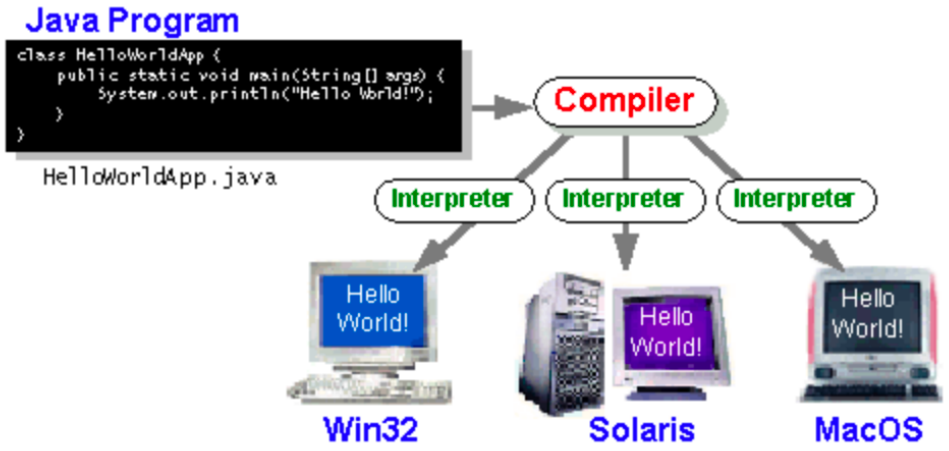
- Grown from a few hundred classes in the JDK (Java Development Kit) to thousands.

- Mascot: Duke



Java: the programming language

- Program is both compiled and interpreted
- Compiler translates program into intermediate platform-independent language
  - Compilation happens once
  - Creates machine instructions for Java Virtual Machine (JVM)
- Interpreter parses and runs each Java bytecode instruction
  - Interpretation happens each time program is executed
  - Interpreter is implementation of JVM



"write once, run anywhere"

Some differences compared to C++

- Mainly for application programming, including web-based and mobile applications
- No operator overloading
- Not really pointers (restricted support, no pointer arithmetic)
- No call by reference
- No destructors
- Automatic garbage collection
- Single class inheritance
- Javadoc (comparable to Doxygen)
- **Basically everything is an object, except fundamental types**
- Also important: keep the class name **the same** as the file name!

Examples…