1 Exercises

Exercise 1. Consider the following *j*-- program:

Greetings.java

```
import java.lang.System;

public class Greetings {
    // Entry point.
    public static void main(String[] args) {
        System.out.println("Hi " + args[0] + "!");
    }
}
```

List the tokens in the program, along with their line numbers and their images.

Exercise 2. Consider a language over the alphabet $\{a, b\}$ that consists of strings ending in ab.

a. Provide a regular expression for the language.

b. Draw a state transition diagram for the language.

Exercise 3. Consider the regular expression $(a|b)*$ over the alphabet $\{a, b\}$.

a. Describe the language implied by the regular expression.

b. Use Thompson's construction to derive a non-deterministic finite state automaton (NFA) recognizing the same language.

c. Use powerset construction to derive an equivalent deterministic finite state automaton (DFA).

d. Use the partitioning method to derive an equivalent minimal DFA.

Exercise 4. Suppose we wish to add support for the `>=` comparison operator in *j*--. What changes will you need to make in the hand-written and JavaCC scanners in the *j*-- code tree in order to support the new operator?

Exercise 5. Suppose we wish to add support for the do-statement in *j*--.

```
statement ::= block
            | DO statement WHILE parExpression SEMI
            | IF parExpression statement [ ELSE statement ]
            | RETURN [ expression ] SEMI
            | SEMI
            | WHILE parExpression statement
            | statementExpression SEMI
```

What changes will you need to make in the hand-written and JavaCC scanners in the *j*-- code tree in order to support the new statement?

2 Solutions

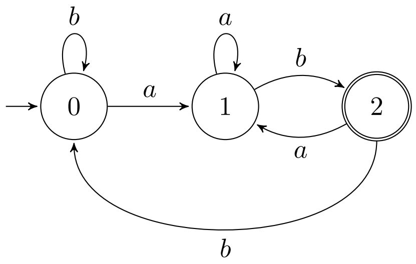
Solution 1.

```
1         : import = import
1         : <IDENTIFIER> = java
1         : . = .
1         : <IDENTIFIER> = lang
1         : . = .
1         : <IDENTIFIER> = System
1         : ; = ;
3         : public = public
3         : class = class
3         : <IDENTIFIER> = Greetings
3         : { = {
4         : public = public
4         : static = static
4         : void = void
4         : <IDENTIFIER> = main
4         : ( = (
4         : <IDENTIFIER> = String
4         : [ = [
4         : ] = ]
4         : <IDENTIFIER> = args
4         : ) = )
4         : { = {
5         : <IDENTIFIER> = System
5         : . = .
5         : <IDENTIFIER> = out
5         : . = .
5         : <IDENTIFIER> = println
5         : ( = (
5         : <STRING_LITERAL> = "Hi "
5         : + = +
5         : <IDENTIFIER> = args
5         : [ = [
5         : <INT_LITERAL> = 0
5         : ] = ]
5         : + = +
5         : <STRING_LITERAL> = "!"
5         : ) = )
5         : ; = ;
6         : } = }
7         : } = }
8         : <EOF> = <EOF>
```

Solution 2.

a. $(a|b) * ab$

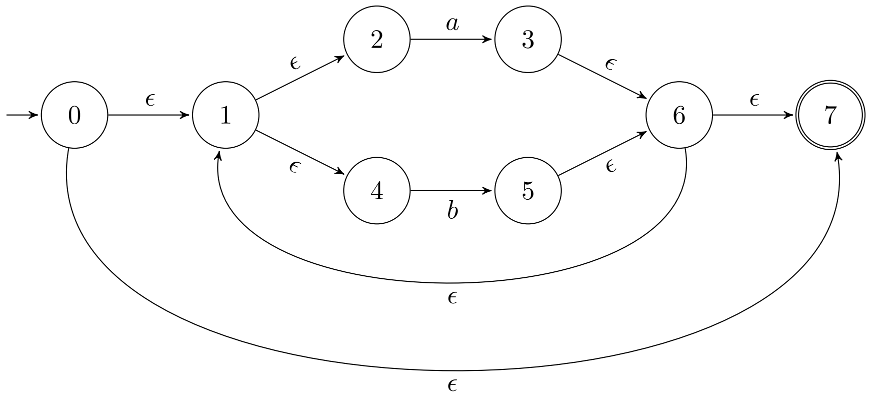
b. State transition diagram for the language:



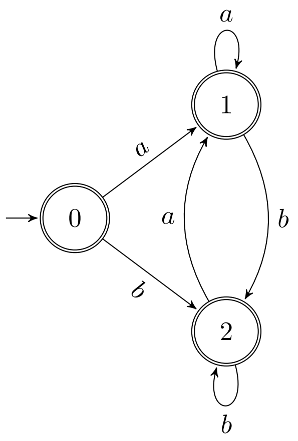
Solution 3.

a. The language consists of strings with any number of a's or b's.

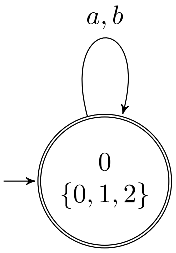
b. An NFA for the language:



c. A DFA for the language:



d. A minimal DFA for the language:



Solution 4.

lexicalgrammar

```
GE ::= ">="
```

TokenInfo.java

```
enum TokenKind {
    GE(">="),
}
```

Scanner.java

```
            case '>':
                nextCh();
                if (ch == '=') {
                    nextCh();
                    return new TokenInfo(GE, line);
                } else {
                    return new TokenInfo(GT, line);
                }
```

j--.jj

```
TOKEN:
{
| < GE: ">=" >
}
```

Solution 5.

lexicalgrammar

```
DO ::= "do"
```

TokenInfo.java

```
enum TokenKind {
    DO("do"),
}
```

Scanner.java

```
reserved.put(DO.image(), DO);
```

j--.jj

```
...
TOKEN:
{
| < DO: "do" >
}
```