

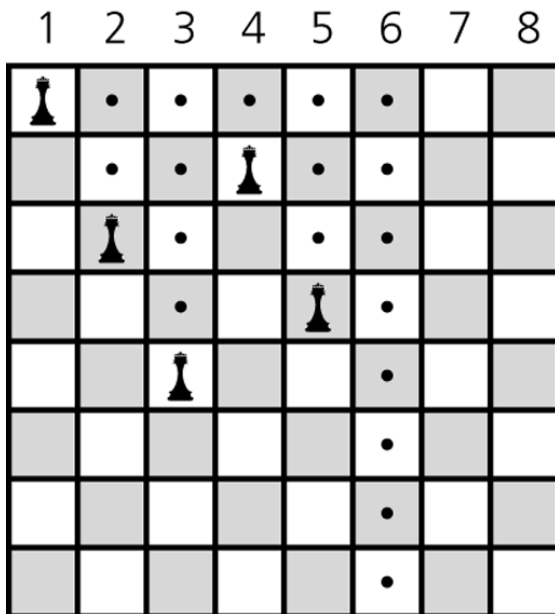
# CHAPTER 5

## Recursion as a Problem-Solving Technique

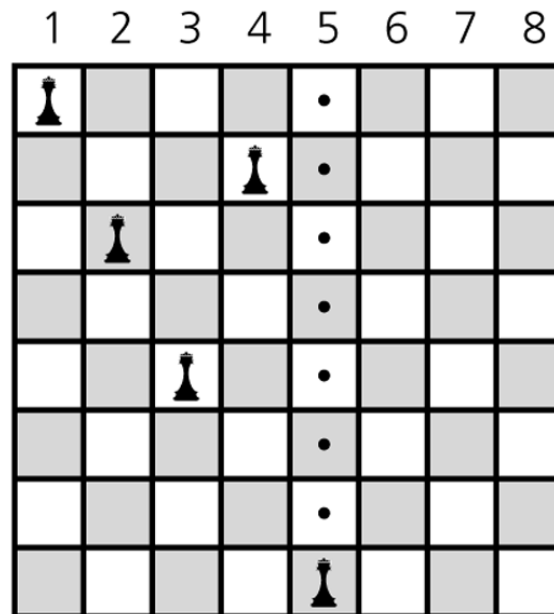
Data Abstraction and Problem Solving with JAVA:  
Walls and Mirrors  
Carrano / Prichard

## Figure 5.1

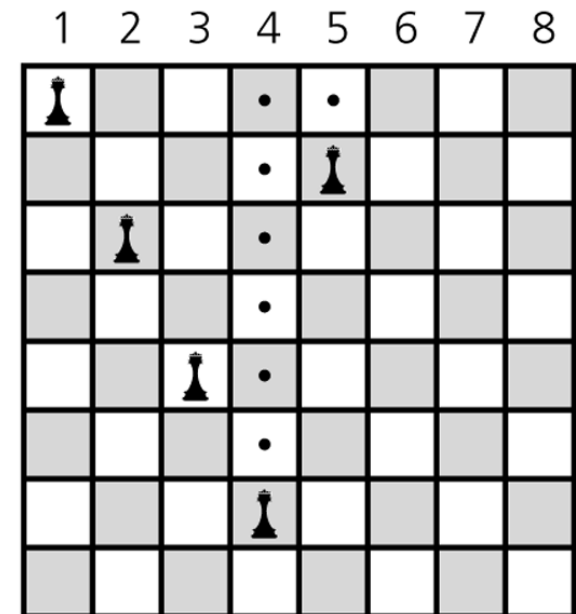
a) Five queens that cannot attack each other, but that can attack all of column 6; b) backtracking to column 5 to try another square for the queen; c) backtracking to column 4 to try another square for the queen and then considering column 5 again



(a)



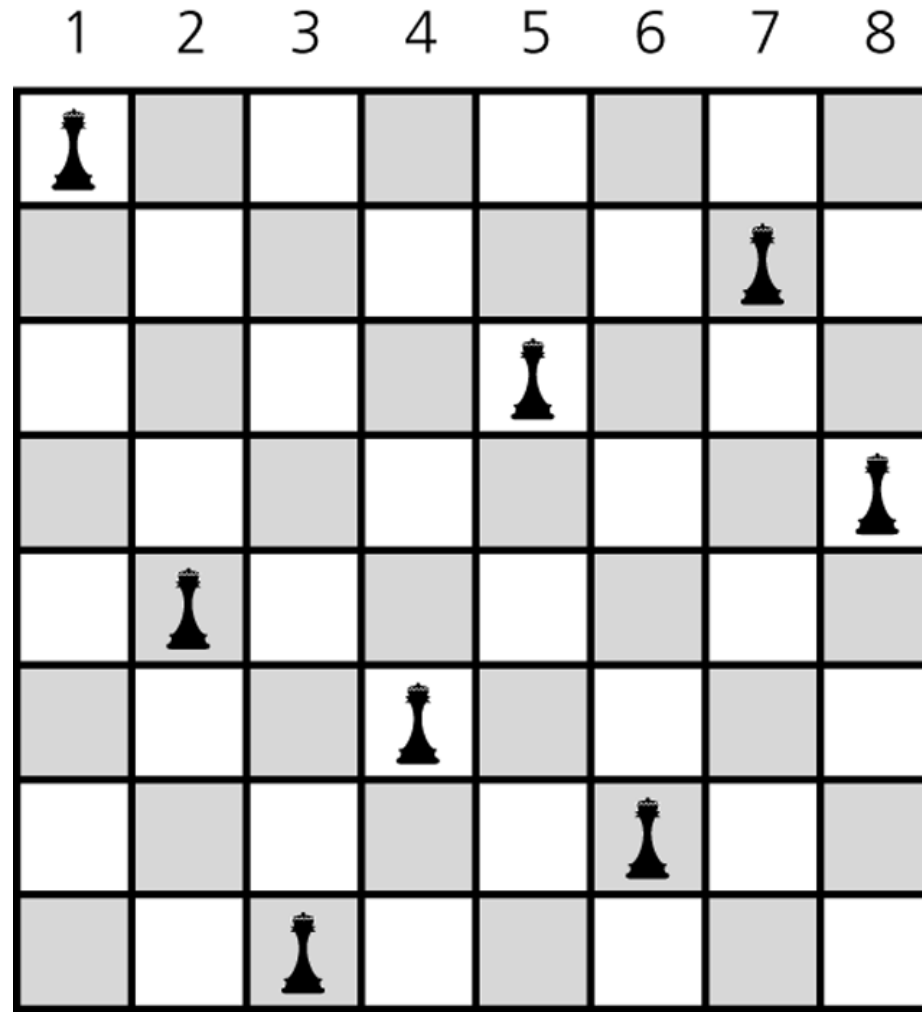
(b)



(c)

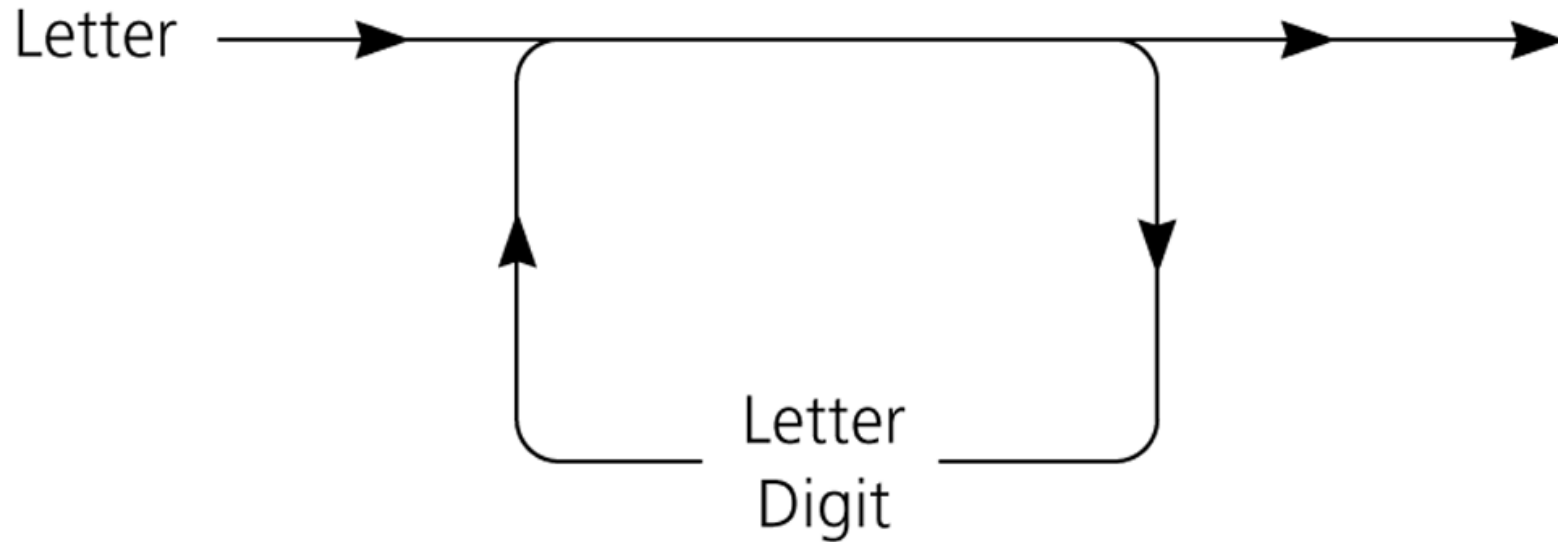
## Figure 5.2

A solution to the Eight Queens problem



## Figure 5.3

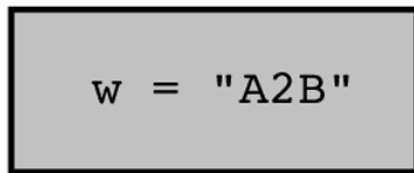
A syntax diagram for Java identifiers



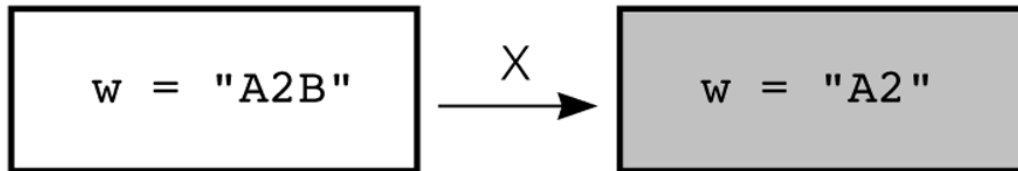
## Figure 5.4a

Trace of `isId("A2B")`

The initial call is made and the method begins execution.



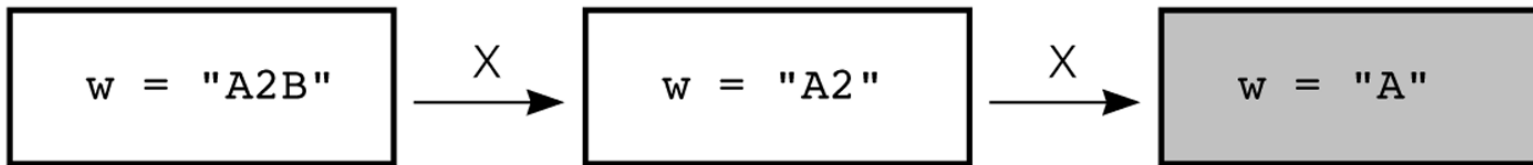
At point X, a recursive call is made and the new invocation of `isId` begins execution:



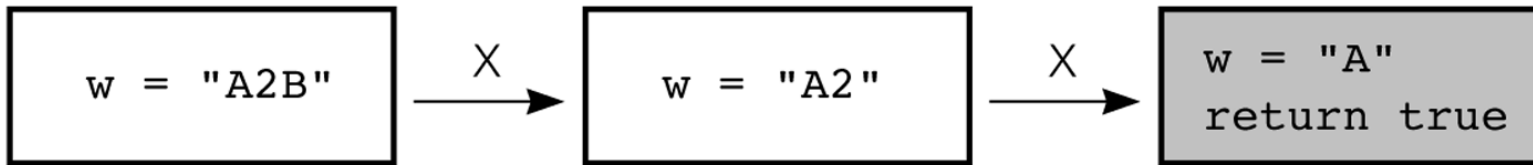
# Figure 5.4b

## Trace of *isId*("A2B")

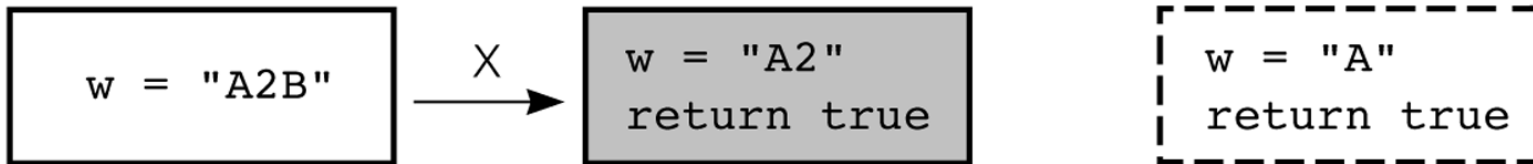
At point X, a recursive call is made and the new invocation of *isId* begins execution:



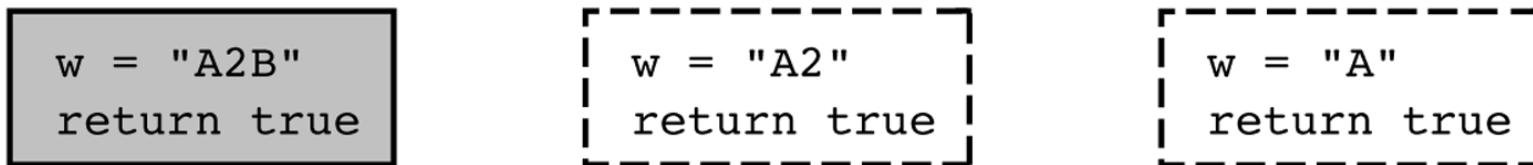
This is the base case, so this invocation of *isId* completes:



The value is returned to the calling method, which completes execution:



The value is returned to the calling method, which completes execution:



# Figure 5.5a

Trace of *endPre*(*first*, *last*), where *strExp* is *+/ab-cd*

The initial call is made and *endPre* begins execution:

```

first      = 0
last      = 6
    
```

first character of *strExp* is +, so at point X, a recursive call is made and the new invocation of *endPre* begins execution:

```

first      = 0
last      = 6
X: endPre(1,6)
    
```

→ X

```

first      = 1
last      = 6
    
```

Next character of *strExp* is /, so at point X, a recursive call is made and the new invocation of *endPre* begins execution:

```

first      = 0
last      = 6
X: endPre(1,6)
    
```

→ X

```

first      = 1
last      = 6
X: endPre(2,6)
    
```

→ X

```

first      = 2
last      = 6
    
```

Next character of *strExp* is a, which is a base case. The current invocation of *endPre* completes execution and returns its value:

```

first      = 0
last      = 6
X: endPre(1,6)
    
```

→ X

```

first      = 1
last      = 6
firstEnd   = 2
    
```

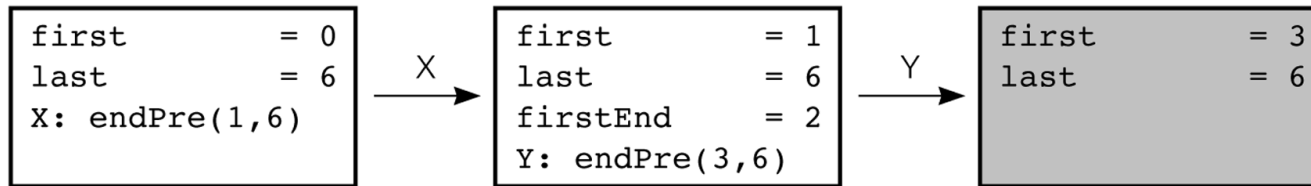
```

first      = 2
last      = 6
return 2
    
```

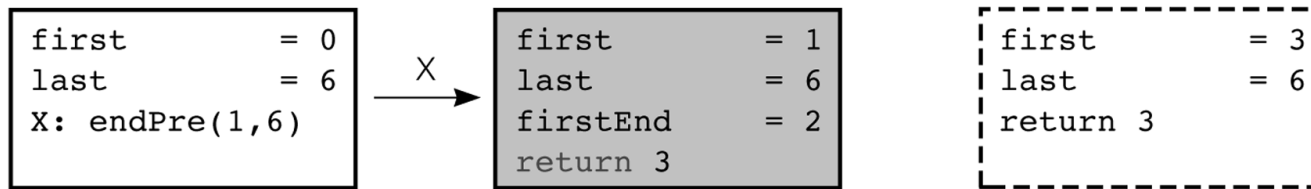
# Figure 5.5b

Trace of *endPre*(*first*, *last*), where *strExp* is *+/ab-cd*

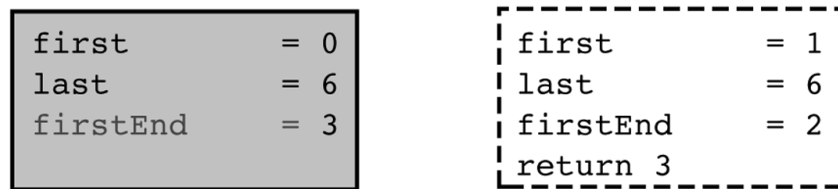
Because *firstEnd* > -1, a recursive call is made from point Y and the new invocation of *endPre* begins execution:



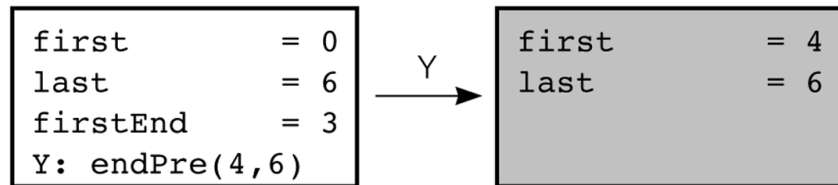
Next character of *strExp* is *b*, which is a base case. The current invocation of *endPre* completes execution and returns its value:



The current invocation of *endPre* completes execution and returns its value:



Because *firstEnd* > -1, a recursive call is made from point Y and the new invocation of *endPre* begins execution:

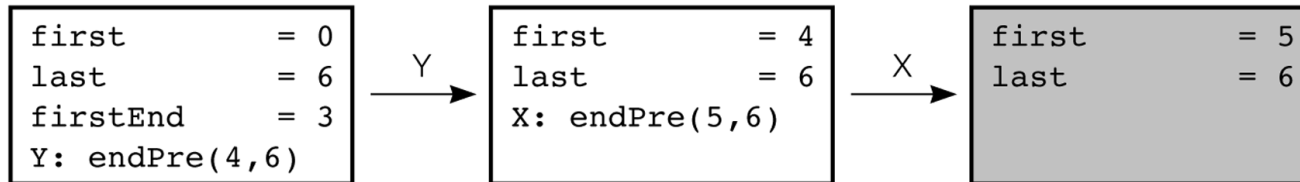




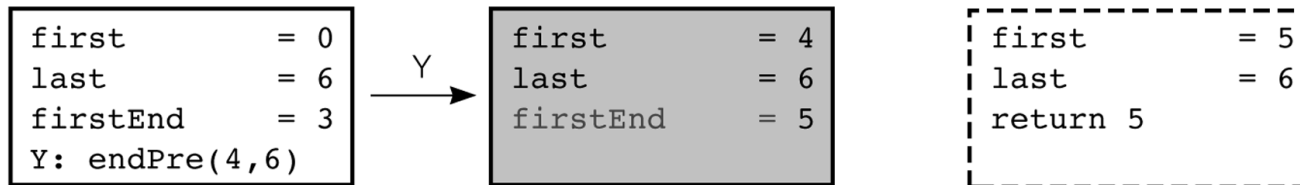
# Figure 5.5c

Trace of *endPre*(*first*, *last*), where *strExp* is *+/ab-cd*

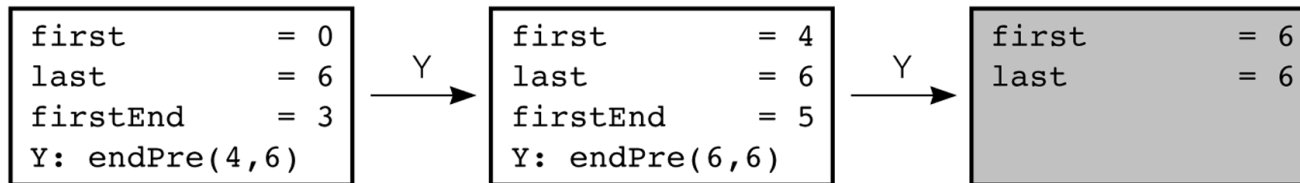
Next character of *strExp* is -, so at point X, a recursive call is made and the new invocation of *endPre* begins execution:



Next character of *strExp* is c, which is a base case. The current invocation of *endPre* completes execution and returns its value:



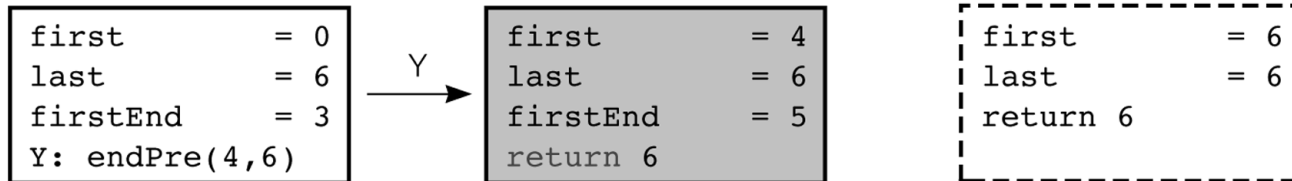
Because *firstEnd* > -1, a recursive call is made from point Y and the new invocation of *endPre* begins execution:



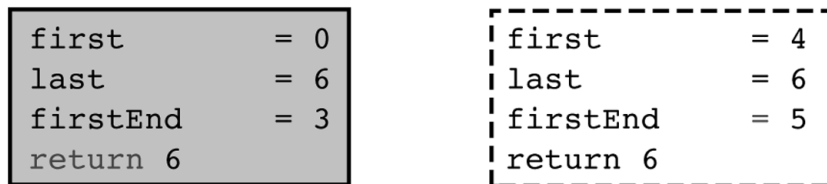
# Figure 5.5d

Trace of *endPre*(*first*, *last*), where *strExp* is *+/ab-cd*

Next character of *strExp* is *d*, which is a base case. The current invocation of *endPre* completes execution and returns its value:



The current invocation of *endPre* completes execution and returns its value:



The current invocation of *endPre* completes execution and returns its value to the original call to *endPre*:

