```
1  // Third Procedural Programming example
2  // contrasting primitive and reference parameters
3
4
5  //import java.util.Arrays;
6
7  class Prog3 {
8
9     // How can we have two methods using the same name (swap)?
10    // because of method _________________
11    private static void swap(int m, int n) {
12       int temp = m;
13       m = n;
14       n = temp;
15    }
16
17    private static void swap(int[] list, int i, int j) {
18       int temp = list[i];
19       list[i] = list[j];
20       list[j] = temp;
21    }
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38    public static void main(String[] args) {
39
40       // Respond to the 4 parts below
41       // ----------------------------
42
43       // 1. Trace the code below to determine output I.a. and I.b. at the console
44       int x = 4, y = 9;
45
46       System.out.println("I.a. : " + x + " "  + y);
47       swap(x, y);
48       System.out.println("I.b. : " + x + " "  + y);
49
50
51
52       // 2. Will the output be different if the formal parameters were named
53       // x and y instead?
54
55
56
57       // 3. Draw the memory diagram as we did last class to show
58       // you understand the underlying mechanism involved in copying
59       // the primitive values passed to the method.
60
61
62
63
64
65
66
67
68
69
70
71
72
73       // 4. You can make the swap work if the values are stored in an array
74
75       // 4.a Write the three lines to define and stores the two values
76       // an array called list (or the one line shorthand version)
77       //int[] list = {x, y};  // shorthand
78
79
80
81
82
83
84
85       //System.out.println("II.a : " + list[0] + " "  + list[1]);
86
87       // 4.b Call the method to swap the two values
88
89
90
91       //System.out.println("II.b : " + list[0] + " "  + list[1]);
92
```

```
 93
 94
 95
 96
 97        // 4.c Try to draw the memory diagram for this array case
 98        // given that for reference the copied value is the address of the object
 99        // (like with Python aliasing)
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135     }
136  }
137
138
139
```