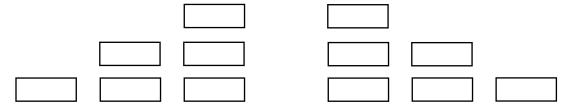
## FRANK CARRANO:

ridini Chiddino

- Recursion is a problem-solving process that breaks a problem into identical but smaller problems.
- Base case: Known solution. Solution stated non-recursively. No recursive call.
- The stack of activation records: Each call to a method generates an activation record that captures the state of the method's execution and that is placed into the program stack. However, these methods need not be distinct. That is, the program stack enables a run-time environment to execute recursive methods. Each invocation of any method produces an activation record that is pushed onto the program stack. The activation record of a recursive method is not special in any way.

Does Java make multiple copies of recursive methods?

No. Java records the current state of the method's execution, including the values of its parameters and local variables as well as the location of the current instructions. Each record is called an activation record and provides a snapshot of a method's state during execution. The records are placed into the program stack. The stack organizes the records chronologically, so that the record of the currently executing method is on top. In this way, Java can suspend the execution of a recursive method and invoked it again with new argument values.



- A recursive method uses more memory than an iterative method, in general, because each recursive call generates an activation record.

```
public class CountDownIterative {
    public static void main(String[] args) {
        int num = 3;
        countDown(num);
    }

    public static void countDown(int x) {
        while (x >= 1) {
            System.out.println("x: " + x);
            x--;
        }
    }
}
```

/\* OUTPUT x: 3 x: 2 x: 1 \*/

- A recursive method that makes many recursive calls will place many activation records in the program stack. Too many recursive calls can use all the memory available for the program stack, making it full. As a result, the error stack overflow occurs. Infinite recursion or large-size problems are the likely causes of this error.
- Tail recursion occurs when the last action of a recursive method is a recursive call. This recursive call performs a repetition that can be done by using iteration.
- Indirect recursion results when a method calls a method that calls a method, and so on until the first method is called again.

```
RECURSION
```

```
64
 65
 66
      public class CountDownRecursive {
                                                                                            /* OUTPUT
 67
           public static void main(String[] args) {
                                                                                           x: 3
 68
               int num = 3;
                                                                                           x: 2
 69
               countDown(num);
                                                                                           x: 1
 70
           }
                                                                                            */
 71
 72
           public static void countDown(int x) {
 73
               System.out.println("x: " + x);
 74
               if (x > 1) {
                                                        // Base case ?
                    countDown(x - 1);
 75
                                                        // Recursive call
 76
 77
           }
 78
      }
 79
 80
 81
 82
      From main
                                                                                   countDown:
 83
                                                                                    x = 1
 84
                                                                                    Base case
 85
 86
 87
 88
                                                           countDown:
                                                                                   countDown:
 89
                                                            x = 2
                                                                                    x = 2
                                                            countDown (1)
 90
 91
 92
 93
                                    countDown:
                                                           countDown:
                                                                                   countDown:
 94
                                                            x = 3
                                    x = 3
                                                                                    x = 3
 95
                                     countDown(2)
 96
 97
 98
 99
            main:
                                    main:
                                                           main:
100
             num = 3
                                                            num = 3
                                    num = 3
                                                                                    num = 3
101
             countDown (3)
                                                             . . .
102
103
104
105
106
                                                                                          Back to main
107
            countDown:
108
             x = 1
             Base case
109
110
111
112
            countDown:
                                    countDown:
113
             x = 2
                                    x = 2
114
115
116
117
118
            countDown:
                                    countDown:
                                                           countDown:
             x = 3
                                                            x = 3
119
                                    x = 3
120
121
122
123
            main:
                                    main:
                                                           main:
                                                                                   main:
124
             num = 3
                                                            num = 3
                                                                                    num = 3
                                    num = 3
125
             . . .
126
```

```
RECURSION
```

```
127
128
129
      public class CountUpRecursive {
                                                                                         /* OUTPUT
130
           public static void main(String[] args) {
                                                                                         x: 1
131
               int num = 3;
                                                                                         x: 2
               countUp(num);
132
                                                                                        x: 3
133
                                                                                         */
134
135
           public static void countUp(int x) {
                                                      // Base case ?
136
               if (x == 1) {
137
                    System.out.println("x: " + x);
138
               } else {
139
                    countUp(x - 1);
                                                   // Recursive call
                    System.out.println("x: " + x);
140
141
               }
142
           }
143
      }
144
145
146
147
                                                                                  countUp:
148
      From main
                                                                                   x = 1
                                                                                   Base case
149
150
151
152
                                                           countUp:
                                                                                  countUp:
153
                                                            x = 2
                                                                                   x = 2
154
                                                            countUp (1)
155
156
157
158
                                   countUp:
                                                           countUp:
                                                                                  countUp:
159
                                    x = 3
                                                            x = 3
                                                                                   x = 3
160
                                    countUp (2)
161
162
163
            main:
                                   main:
                                                           main:
                                                                                  main:
164
                                                            num = 3
             num = 3
                                    num = 3
                                                                                   num = 3
165
             countUp(3)
                                                            . . .
166
167
168
169
                                                                                         Back to main
            countUp:
170
             x = 1
171
             Base case
172
173
174
175
            countUp:
                                   countUp:
176
             x = 2
                                    x = 2
177
178
179
180
181
            countUp:
                                   countUp:
                                                           countUp:
             x = 3
                                    x = 3
                                                            x = 3
182
183
184
185
186
            main:
                                   main:
                                                           main:
                                                                                  main:
187
                                                            num = 3
             num = 3
                                    num = 3
                                                                                   num = 3
188
             . . .
                                                            . . .
189
```

```
191
192
      public class SumOf {
                                                                                       /* OUTPUT
193
          public static void main(String[] args) {
                                                                                       465
194
               int x = 30;
                                                                                       */
195
               System.out.println(sumOf(x));
196
          }
197
198
          public static int sumOf(int n) {
199
               int sum;
               if (n == 1) {
200
                                                    // Base case ?
201
                   sum = 1;
202
               } else {
203
                   sum = sumOf(n - 1) + n;
                                                     // Recursive call
204
               }
205
               return sum;
206
          }
207
      }
                                                  n
208
209
210
211
212
213
                                                  n
214
215
216
                                                  n
217
218
219
                                                  n
220
221
222
223
                                                  n
224
225
226
                                                  n
227
228
229
                                                  n
230
231
232
                                                  n
233
234
235
                                                  n
236
237
238
239
240
241
```

## DESIGN GUIDELINES FOR SUCCESSFUL RECURSION

- The method must be given an input value, usually as an argument
- The method definition must contain logic that involves this input value and leads to different cases. Typically, such logic includes an if statement or a switch statement.
- One or more of these cases should provide a solution that does not require recursion. There are the base cases or stopping cases.
- One or more cases must include a recursive invocation of the method. These recursive invocations should in some sense take a step toward a base case by using "smaller" arguments or solving "smaller" versions of the task performed by the method.

```
254
255
     public class DisplayArray {
256
257
         public static void main(String[] args) {
258
              int[] nums = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21};
259
              displayArrayFirst(nums, 0, 5);
             System.out.println("----");
260
             displayArrayLast(nums, 0, 5);
261
262
             System.out.println("----");
263
             displayArrayHalf(nums, 0, 5);
264
265
             System.out.println("----");
266
              int reviewMid = 3;
267
             System.out.println("reviewMid: 3/2 = " + reviewMid / 2);
268
          }
269
270
          // The first entry is printed at the first method call.
271
         public static void displayArrayFirst(int array[], int first, int last) {
272
              System.out.println("First: " + first + " | Entry: " + array[first] + " ");
273
              if (first < last) {</pre>
274
                 displayArrayFirst(array, first + 1, last);
275
              }
276
          }
277
278
          // The first entry is printed when the base case is reached.
279
          // The others are suspended in the stack.
280
         public static void displayArrayLast(int array[], int first, int last) {
281
             if (first <= last) {</pre>
282
                 displayArrayLast(array, first, last - 1);
                  System.out.println("Last: " + last + " | Entry: " + array[last] + " ");
283
284
              }
285
          }
286
287
         public static void displayArrayHalf(int array[], int first, int last) {
288
              if (first == last) { // Base case: array of 1 element
289
                  System.out.println("First|Last: " + first + "|" + last + " \t| Entry: " +
290
      array[first] + " ");
291
              } else { // If array has more than 1 element, keep dividing
292
                  int mid = (first + last) / 2;
293
                  System.out.print(" \t\t\t\t\t| Mid: " + mid + "\n");
294
                  displayArrayHalf(array, first, mid);
295
                 displayArrayHalf(array, mid + 1, last);
296
              }
297
          }
298
      }
299
300
     First: 0 | Entry: 1
301
     First: 1 | Entry: 3
302
     First: 2 | Entry: 5
                                                                       | Mid: 2
303
     First: 3 | Entry: 7
                                                                       | Mid: 1
304
     First: 4 | Entry: 9
                                                                       | Mid: 0
305
     First: 5 | Entry: 11
                                   First|Last: 0|0
                                                     | Entry: 1
306
      ______
                                   First|Last: 1|1
                                                     | Entry: 3
307
     Last: 0 | Entry: 1
                                   First|Last: 2|2
                                                     | Entry: 5
308
     Last: 1 | Entry: 3
                                                                       | Mid: 4
309
     Last: 2 | Entry: 5
                                                                       | Mid: 3
310
     Last: 3 | Entry: 7
                                   First|Last: 3|3
                                                     | Entry: 7
311
     Last: 4 | Entry: 9
                                   First|Last: 4|4
                                                     | Entry: 9
312
     Last: 5 | Entry: 11
                                   First|Last: 5|5
                                                     | Entry: 11
313
                                   ______
314
                                   reviewMid: 3/2 = 1
315
```

```
317
318
      public class FibonacciTimed {
319
320
          public static void main(String args[]) {
321
322
              int n = 40;
323
324
              long start = System.nanoTime();
325
              System.out.println("Recursive:");
326
              for (int i = 0; i \le n; i++) {
327
                  System.out.print(fibonacciRecursive(i) + " ");
328
              }
329
              long end = System.nanoTime();
              System.out.format("\nTime: %,d \ microseconds\n", \ (end - start) / 1000);
330
331
332
              start = System.nanoTime();
333
              System.out.println("\nIterative:");
334
              for (int i = 0; i <= n; i++) {
335
                  System.out.print(fibonacciIterative(i) + " ");
336
              }
337
              end = System.nanoTime();
338
              System.out.format("\nTime: %,d microseconds\n", (end - start) / 1000);
339
340
              System.out.println("");
          }
341
342
343
          public static int fibonacciRecursive(int n) {
344
              if (n \le 1) {
345
                  return n;
346
              }
347
              return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);
348
          }
349
350
          public static int fibonacciIterative(int n) {
351
352
              if (n \le 1) {
353
                  return n;
354
              }
355
              int fib = 1;
              int prevFib = 1;
356
357
              for (int i = 2; i < n; i++) {
358
359
                  int temp = fib;
360
                  fib += prevFib;
361
                  prevFib = temp;
362
363
              return fib;
364
          }
365
366
      }
367
368
      Recursive:
369
      0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711
370
      28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578
371
      5702887 9227465 14930352 24157817 39088169 63245986 102334155
372
      Time: 1,198,879 microseconds
373
374
      Iterative:
      0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711
375
376
      28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578
377
      5702887 9227465 14930352 24157817 39088169 63245986 102334155
378
      Time: 786 microseconds
```