Problem: To find the sum of all even Fibonacci numbers less than Fn for some given n.

There is no real motivation for this problem, but it turns out to be interesting to analyze.

Naive approach:

```
long sum = 0;
for (int k = 1;  k <= n; k++)
{
  long next = fib(k);
  if (next % 2 == 0)
    sum += next;
}
```

Case 1: fib(k) is implemented recursively.

In this case, the execution time of fib is given by $T(n) = T(n-1) + T(n-2) + k$. This is dominated by $2T(n-1) + k$ and dominates $2T(n-2)+ k$. These have asymptotic behaviour 2^n and $\sqrt{2}^n$ respectively and thus we can say that fib(n) is $O(2^n)$ making the whole algorithm $O(2^n)$. Yuck!

An obvious improvement is given by:

Case 2: fib(k) is implemented iteratively. In this case, fib(k) is $O(k)$, so the overall complexity is $O(n^2)$.

Two approaches to further improvement are given by

Case 3a: Calculate the sum in the same loop where the Fibonacci numbers are calculated. This de-structures the code, but results in complexity $O(n)$.

Better, I think is

Case 3b: implement fib(k) recursively but using dynamic programming *via* memoization. This makes fib(n) O(c) at the expense of increasing the space to O(n). Again, the overall complexity becomes O(n).

At this point, we should probably be satisfied, but it gets better!

First, a little preliminary work:

Claim: $f(0) + f(1) + f(2) + \ldots + f(n) = f(n) + f(n+1) - 1$
Proof:
This is clearly true when $n = 0$, for the formula gives $0 + 1 - 1 = 0$.
Let $n = k$. Then by the inductive hypothesis we have
$f(0) + f(1) + \ldots + f(k) = f(k) + f(k+1) - 1$. Adding $f(k+1)$ to each side gives $f(0) + f(1) + \ldots + f(k) + f(k+1) = [f(k) + f(k+1)]+ f(k+1) - 1$. But $f(k) + f(k+1) = f(k+2)$, so the right hand side becomes $f(k+1) + f(k+2) - 1$, which completes the proof.

Claim: for any non-negative integer n, f(3n) is an even number, f(3n+1) and f(3n+2) are odd numbers.
Proof:
$f(0) = 0$ which is even. $F(1) = f(2) = 1$ and thus both are odd.

Suppose f(3k) is an even number, and f(3k+1) and f(3k+2) are odd.
Then f(3k+3) is the sum of two odd numbers and thus even. f(3k+4) is the sum of an odd number (f(3k+2)) and an even number (f(3k+3)) and thus odd. f(3k+5) is also the sum of an odd (f(3k+4)) and an even (f(3k+3)) so it too is odd. This completes the proof.

Finally, consider the sum
$f(0) + [f(1) + f(2)] + f(3) + [f(4) + f(5)] + f(6) +...+ f(3n-2) + f(3n-1)] + f(3n)$

This, as observed above is equal to $f(3n) + f(3n+1) -1$.

But each of the bracketed pairs sum to the succeeding Fibonacci number, so this sum becomes.

f(3) + f(3) + f(6) + f(6) + f(9) + f(9) + … + f(3n) + f(3n)
= 2f(3) + 2f(6) + 2f(9) … + 2f(3n) = f(3n) + f(3n+1) – 1.
The left hand side is just twice the sum of the even Fibonacci numbers less than or equal to 3n. Thus, the sum we seek is given by

[f(3n) + f(3n+1)]/2.

Finally, it is well known that f(n) can be expressed as
[Phi^n -(-Phi)^-n]/sqrt(5) where Phi is the so called golden ratio.

Since 1/Phi is approximately 0.612, the second term is less than ½ for any n greater than 1. This means f(n) is given by round(Phi^n)/sqrt(5).

This gives us our final algorithm:

```
public static long sumEven(int n)
{
  final double PHI = (1 + Math.sqrt(5.0) / 2.0;
  while(n % 3 != 0)
    n--;
  long f3n = (long)Math.round(raise(PHI, n));
  long f3np1 = (long)Math.round(raise(PHI, n+1));
  return (f3n + f3np1 -1)/2;

}
```

raise(base, exp) can be written recursively to be O(lg(n)), which gives us a lg(n) solution to the original problem. Not bad improvement for a problem where we started at 2^n.