Given:

$g(x) = \cos(x) - x^3$ {has root on (0, 1)}

Floating Point ~ 0.60352

Find answer within 4 decimal places; Satisfy $|x_c - r| < 0.5 \times 10^{-4}$

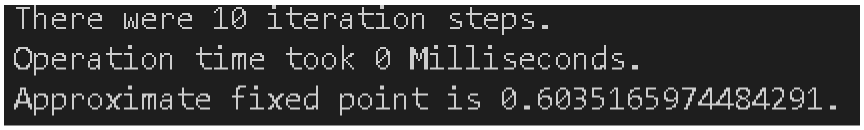
Problem Statement 4: Use fixed-point iterations to find a fixed-point starting from $x_0 = 0$ for $g(x)$.

Algorithm Description: The floating point x is such that $x = g(x)$, where the following algorithm considers if the output resulting from x is equal to x itself to an acceptable level of tolerance defined in the problem. If the output is not equal to x, then we simply use the output as the input to itself until an acceptable floating point is discovered. The original way the equation $g(x) = x = \cos(x) - x^3$ was written did not converge to a floating point and instead diverged, so I changed up the equation by turning $-x^3$ into $-3x^3+2x^3$ so I could get the equation in a form that converged, which it did by rewriting as $x + 3x^3 = \cos(x) + 2x^3$ which was just $x = (\cos(x) + 2x^3) / (1 + 3x^2)$.

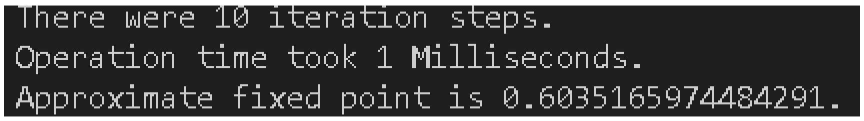
Pseudocode:

```
Function fixedPointIterations(x, previousSoln, count):
    If abs(x - previousSoln) < tolerance AND count > 1:
        print("There were " + count + " iterations.")
        Return x

    Return fixedPointIterations(g(x), x, count + 1);
End

Function g(x):
    Return $(\cos(x) + 2x^3) / (1 + 3x^2)$;
End
```

Results: (p14.java)



```
There were 10 iteration steps.
Operation time took 0 Milliseconds.
Approximate fixed point is 0.6035165974484291.
```



```
There were 10 iteration steps.
Operation time took 1 Milliseconds.
Approximate fixed point is 0.6035165974484291.
```

Observations & Analysis

The root estimated by *p14.java* is approximately 0.603517 which is well within 4 decimal points of the approximately correct root of 0.60352. As for the program efficiency, I would say it is sufficient as it only took no more than a millisecond to conduct 10 iterations of 11 floating point operations at most (last iteration could be fewer operations returning on first if statement) on each iteration which is about at least 110,000 operations per second. The algorithm is good because it only takes no more than 1 millisecond to figure out the root. I am guessing there are ways to write algorithms using bitwise operators for probably faster results and other more ways unknown to me that could result in faster runtimes. This program is scalable to any predefined initial point generally and runs with any java configured environment.
