**Code Explanation**

**1. Class Structure & Overview**

File Name: `StockMagic.java`

Class Name: `StockMagic`

Purpose: Analyses stock prices using four key operations while maintaining a playful theme with method names and emojis.

**2. Methods & Functionality**

2.1 `calculateAverage(float[] prices)`

* Purpose: Computes the average stock price.
* Mechanics:
* Iterates through the `prices` array using a for-each loop.
* Accumulates the total sum of all prices.
* Divides the total by the number of prices.
* Example:

For input `[10.0f, 20.0f, 30.0f]`, returns `20.0f`.

2.2 `findMaximum(float[] prices)`

* Purpose: Find the highest stock price.
* Mechanics:
* Initialises `max` with the first element of the array.
* Compares each subsequent price to update `max`.
* Example:

For input `[15.0f, 25.0f, 10.0f]`, returns `25.0f`.

2.3 `countPrices(float[] prices, float target)`

* Purpose: Counts occurrences of a specific price.
* Mechanics:
* Uses a for-each loop to check each price against `target`.
* Increments `count` for exact matches.
* Example:

For input `[5.0f, 5.0f, 10.0f]` and target `5.0f`, returns `2`.

2.4 `cumulativeSum(ArrayList<Float> prices)`

* Purpose: Generates cumulative sums of stock prices.
* Mechanics:
* Maintains a running total (`sum`).
* Adds the cumulative value at each step to the result list.
* Example:

For input `[1.0f, 2.0f, 3.0f]`, returns `[1.0f, 3.0f, 6.0f]`.

**3. `main` Method Execution Flow**

1. Data Initialisation:

float[] prices = {45.8f, 43.2f, 44.9f, 48.3f, 47.1f, 49.7f, 46.4f, 50.2f, 49.9f, 48.6f};

ArrayList<Float> priceList = new ArrayList<>();

for(float p : prices) priceList.add(p); // Convert array to ArrayList

2. Method Calls & Output:

System.out.println("✨ Average: " + calculateAverage(prices));

System.out.println("🐉 Maximum: " + findMaximum(prices));

System.out.println("🔍 Count of 48.3: " + countPrices(prices, 48.3f));

System.out.println("📈 Cumulative Sums: " + cumulativeSum(priceList));

**4. Key Concepts Used**

* For-Each Loops: Simplify iteration over arrays/collections.
* Array vs. ArrayList: Demonstrates conversion between the two.
* Static Methods: Called directly without object instantiation.
* Floating-Point Precision: Uses `float` for decimal values (note the `f` suffix).

**5. Sample Output:**

Average: 47.41

Maximum: 50.2

Count of 48.3: 1

Cumulative Sums: [45.8, 89.0, 133.9, 182.2, 229.3, 279.0, 325.4, 375.6, 425.5, 474.1]

**6. Best Practices Illustrated**

1. **Thematic Naming:** Methods like `crystalBallAverage` and `dragonMaxTreasure` improve readability.
2. **Separation of Concerns:** Each method handles one specific task.
3. **Emoji Visualisation:** Output uses symbols for clarity.
4. **Input Validation:** Assumes valid input as per problem constraints.

**7. Edge Cases & Notes**

* Empty Input: Will throw errors (handled by problem constraints).
* Floating-Point Comparison: `countPrices` uses exact equality (`==`), which may miss values with minor precision differences.
* Performance: All methods run in O (n) time, efficient for large datasets.