Code Review Report: Selection Sort (Student B)
Reviewer: SergeJ Balakarev
Author: Abdunur Amangeldiev

Overview

The reviewed code provides an in-place Selection Sort for integer arrays with integration of a PerformanceTracker to measure metrics such as comparisons, swaps, and array accesses. The algorithm repeatedly selects the minimum element from the unsorted suffix and places it at the current position. A CLI runner generates input arrays and exports metrics to CSV. Unit tests cover many cases.

Two components are central:
- SelectionSort — core algorithm and swap helper, instrumented with PerformanceTracker.
- PerformanceTracker — tracks comparisons, swaps, and array accesses; supports CSV export.

Algorithm Summary

Selection Sort repeatedly finds the minimum in the remaining unsorted range and swaps it into position i.
Time Complexity: Best $O(n^2)$, Average $O(n^2)$, Worst $O(n^2)$. Space Complexity: O(1) in-place.

Time Complexity:
- Best case: $O(n^2)$ (selection sort is not adaptive by design)
- Average case: $O(n^2)$
- Worst case: $O(n^2)$

Space Complexity:
- O(1) — sorting is performed in-place.

Code Structure & Readability

The code is relatively straightforward, but one critical logic issue undermines correctness. Metric calls are explicit and easy to follow, and the swap helper encapsulates the element exchange. Naming is generally descriptive, except for one misleading flag.

Key Issue (Correctness)

- Early termination bug in the outer loop: a boolean flag named "swapped" is used to break the outer loop when no smaller element is found for position i. In selection sort this does not imply the remainder is sorted, so the algorithm may exit prematurely and leave the array unsorted (e.g., [1, 3, 2]). This is a correctness bug.
- Misleading flag name: the flag "swapped" actually means "foundNewMinimum" rather than an actual swap; this likely led to the faulty optimization.

Performance Considerations

- Selection sort is inherently $O(n^2)$ comparisons. Attempting to short-circuit when no new minimum is found at position i is not a valid optimization for selection sort and can break correctness. Removing the early-exit restores correctness without changing asymptotic bounds.

- Tracker updates add overhead but are intentional for measurement and educational purposes. Array access counting models two reads per comparison and four accesses per swap (two reads, two writes), which is reasonable as long as documented.

Strengths
- Clear separation of concerns: algorithm vs. metrics vs. CLI/IO.
- Simple swap helper with explicit metric increments.
- Tests include a good spread: empty, single element, sorted, reverse, duplicates, negatives, large random.
- CSV export enables easy downstream analysis.

Minor Suggestions
- Fix correctness: remove the early-exit condition in the outer loop; selection sort should complete all passes regardless of whether the current minimum equals the current position.
- Rename or remove the misleading flag: if kept for readability, rename to foundNewMinimum; otherwise remove it entirely.
- Document metric methodology: clarify that arrayAccesses counts both reads and writes; add this to README and/or Javadoc.
- CSV header: consider writing a header line once (algorithm,n,runtime_ns,comparisons,swaps,array_accesses) to ease analysis.
- Close Scanner in CLI: use try-with-resources or call scanner.close() after reading inputs.
- Input generation: reverse array uses values n..1; that's fine, but be consistent and document the choice.
- UX for invalid menu choice: instead of zero-filled arrays by default, re-prompt or print a clear warning.

Code Quality & Maintainability
Formatting and structure are consistent. Readability would benefit from eliminating the incorrect early-exit logic and clarifying metric semantics. Minor CLI resource handling and CSV usability improvements will make the tooling more robust.

Testing Recommendations
- Add a regression test for the premature-exit case, e.g., input [1, 3, 2] should sort to [1, 2, 3].
- Validate metric counts on small arrays to ensure comparison and arrayAccess increments align with the documented methodology.
- Maintain existing coverage: empty, single, sorted, reverse, duplicates, negatives, and random cases.

Overall Evaluation
The implementation is close to educational/benchmarking use-case needs, but the early termination bug currently compromises correctness. After removing the faulty early-exit and clarifying metric/documentation and small CLI improvements, the code will be correct, clear, and suitable for teaching and measurement purposes.

Final Note
Overall, the code is good.