

Solutions

0.1 Design Before Implementation

Reference example

Input: nonempty list of scores

Output: largest score

Rule: do not modify the input

Cases: one item, duplicates, negative values

0.1.1 - Explain

A maximum finder starts best at 0. Give a failing input and explain a correct initialization.

For [-8, -3], it incorrectly returns 0. For a nonempty input, initialize best to the first element, then compare the remaining values.

0.1.2 - Write code

Implement and demonstrate a maximum scan for the nonempty array {-8, -3, -10} without changing it.

Initialize from the input and maintain the largest value seen.

```
int[] scores = {-8, -3, -10};  
int best = scores[0];  
for (int value : scores) if (value > best) best = value;  
System.out.println(best);
```

Expected output

-3

Solutions

0.2 Abstraction and Encapsulation

Reference example

```
class Meter {
    private int value;
    void add(int amount) { if (amount > 0) value += amount; }
    int read() { return value; }
}
Meter m = new Meter();
m.add(4); m.add(-2);
System.out.println(m.read());
```

0.2.1 - Explain

Why can returning an internal mutable list from an accessor weaken encapsulation?

The caller may alter the same list and bypass the class validation. Return an appropriate defensive copy or controlled view when independent mutation must not be allowed.

0.2.2 - Write code

Create a Counter with a private count and an increment method, then show two increments.

Keep mutation inside the class.

```
class Counter {
    private int count;
    void increment() { count++; }
    int getCount() { return count; }
}
Counter c = new Counter(); c.increment(); c.increment();
System.out.println(c.getCount());
```

Expected output

2

Solutions

0.3 Testing and Debugging Workflow

Reference example

```
int[] values = {-4, -1, -9};
int best = values[0];
for (int value : values) best = Math.max(best, value);
System.out.println(best);
```

0.3.1 - Trace

Determine the exact output and explain the state changes.

```
int[] a = {1, 2, 3};
int sum = 0;
for (int i = 0; i < a.length - 1; i++) sum += a[i];
System.out.println(sum);
```

Output

3

The final index is excluded. A full traversal should use $i < a.length$.

0.3.2 - Explain

Give boundary tests for a method counting positive array elements.

Test an empty array, a single positive value, a single zero, all negatives, and mixed values. Expected counts include 0 for empty and zero-only input.

Solutions

0.4 Technical Communication

Reference example

Claim: a full scan of n elements uses $O(n)$ time

Evidence: exactly n loop iterations

Space: $O(1)$ auxiliary storage

Precondition: input array is available

0.4.1 - Explain

Rewrite "this algorithm is fast" as a precise claim for a loop that visits n elements once.

It takes linear time, $O(n)$, under a constant-cost per-element operation model. State whether the claim is worst-case and what counts as input size.

0.4.2 - Explain

Why should a bug report include input, expected output, actual output, and reproduction steps?

Those details let another person reproduce the failure and distinguish a program defect from a misunderstanding of the requirement.