Exercise walk-through

4.7 Wrapper Classes

AP Computer Science A

The five standard array algorithms

sum or average	an accumulator declared outside the loop; divide by `length` at the end
max or min	start from `arr[0]`, not from zero – an array of negatives has no element above zero
count matching	a counter, incremented only when the element passes the test
check for a duplicate	a nested loop, with the inner one starting at `i+1` so no pair is checked twice
reverse or copy	build into a new array, or swap ends inward for an in-place reverse

> Every one of these is a traversal plus a small body. Recognising which one a question wants is most of the work.

Why a data structure at all

the problem	a single variable holds one value; real problems need many related values – a class roster, pixels, sensor readings
a data structure	holds many values under one name, reached by index
an array	fixed length, set when it is created, and `arr.length` is a field, not a method
the bounds	valid indices run 0 to `length-1`; anything outside throws an `ArrayIndexOutOfBoundsException`
the run-time cost	reaching any element takes the same time whatever the index – that is the point of an array

> Off-by-one at the top end is the classic array bug: `i <= arr.length` throws, `i < arr.length` does not.

Removing from a list while looping over it

the problem	`remove(i)` shifts every later element down, so the element after the removed one is skipped
fix one	loop backwards, from `size()-1` down to `0` – later shifts cannot affect indices you have passed
fix two	do not increment `i` after a removal, so the next pass re-examines the same index
never	remove inside an enhanced `for` loop – it throws a `ConcurrentModificationException`
the rest	the array algorithms carry over unchanged, plus insertion and deletion, which arrays cannot do easily

> This is the single most-tested ArrayList point. If a question removes items in a loop, it is testing exactly this.

Exercises 2.1·2.3

2.1 Explain why `ArrayList<int>` does not compile. [1]

2.2 State whether autoboxing or unboxing happens in `double d = prices.get(2);`, where `prices` is an `ArrayList<Double>`. [1]

2.3 State the value of `Integer.parseInt("12") + Integer.parseInt("3")` and of `"12" + "3"`. [2]

Exercises 2.1·2.3

2.1 Explain why `ArrayList<int>` does not compile. [1]

2.1

- an `ArrayList` holds **objects** only; the type must be the wrapper, `Integer`

2.2 State whether autoboxing or unboxing happens in `double d = prices.get(2);`, where `prices` is an `ArrayList<Double>`. [1]

2.2

- **unboxing**: the `Double` from `get` becomes a `double`

2.3 State the value of `Integer.parseInt("12") + Integer.parseInt("3")` and of `"12" + "3"`. [2]

2.3

- `12 + 3` = **15**
- string concatenation: **"123"**

Exercises 2.4·3.1

2.4 State the value of `Integer.MIN_VALUE - 1`. [1]

3.1 Which statement causes **unboxing**? [1]

- **A** `Integer x = 4;`
- **B** `list.add(10);`
- **C** `int y = list.get(1);`
- **D** `Double z = 2.5;`

Exercises 2.4·3.1

2.4 State the value of `Integer.MIN_VALUE - 1`. [1]

2.4

- it overflows to **`Integer.MAX_VALUE`**, $2^{31} - 1$

3.1 Which statement causes **unboxing**? [1]

☐ **A** `Integer x = 4;`
B `list.add(10);`
C `int y = list.get(1);`
D `Double z = 2.5;`

3.1 C

- `get` returns an `Integer`, stored in an `int`

Exercise 3.2

3.2 `nums` is an `ArrayList<Integer>` holding `[3, null, 8]`. What happens? [1]

```
int total = 0;
for (Integer n : nums) total += n;
```

- [ ] **A** `total` is 11
- [ ] **B** `total` is 3
- [ ] **C** a `NullPointerException` is thrown
- [ ] **D** it does not compile

Exercise 3.2

3.2 `nums` is an `ArrayList<Integer>` holding `[3, null, 8]`. What happens? [1]

```
int total = 0;
for (Integer n : nums) total += n;
```

- ☐ **A** `total` is 11
- ☐ **B** `total` is 3
- ☐ **C** a `NullPointerException` is thrown
- ☐ **D** it does not compile

3.2 C

- `total += n` unboxes `null`

Exercise 3.3

3.3 `best` should hold the largest value of `int[] arr`, which may hold only negative numbers. Which start value is always correct? [1]

- **A** `int best = 0;`
- **B** `int best = Integer.MIN_VALUE;`
- **C** `int best = Integer.MAX_VALUE;`
- **D** `int best = -1;`

Exercise 3.3

3.3 `best` should hold the largest value of `int[] arr`, which may hold only negative numbers. Which start value is always correct? [1]

- **A** `int best = 0;`
- **B** `int best = Integer.MIN_VALUE;`
- **C** `int best = Integer.MAX_VALUE;`
- **D** `int best = -1;`

3.3 B

- 0 and -1 are larger than every value of an all-negative array

Exercise 3.4 (a)

3.4 A quiz app stores answers typed by players as `String`s in `ArrayList<String> typed`, such as `["12", "7", "30"]`.

(a) Write a code segment that builds `ArrayList<Integer> values` from `typed`, converting each `String` with `Integer.parseInt`. [3]

Exercise 3.4 (a)

3.4 A quiz app stores answers typed by players as `Strings` in `ArrayList<String> typed`, such as `["12", "7", "30"]`.

(a) Write a code segment that builds `ArrayList<Integer> values` from `typed`, converting each `String` with `Integer.parseInt`. [3]

(a)

```
ArrayList<Integer> values = new ArrayList<Integer>();
for (String s : typed)
{
    values.add(Integer.parseInt(s));
}
```

- creates an empty `ArrayList<Integer>`
- traverses every `String` in `typed`
- adds `Integer.parseInt(s)` for each

Exercise 3.4 (b)

3.4 A quiz app stores answers typed by players as `Strings` in `ArrayList<String> typed`, such as `["12", "7", "30"]`.

(b) Identify where autoboxing happens in your code. [1]

Exercise 3.4 (b)

3.4 A quiz app stores answers typed by players as `Strings` in `ArrayList<String> typed`, such as `["12", "7", "30"]`.

(b) Identify where autoboxing happens in your code. [1]

(b)

- in `values.add(…)`: the `int` from `parseInt` is boxed to an `Integer`

Exercise 3.4 (c)

3.4 A quiz app stores answers typed by players as `String`s in `ArrayList<String> typed`, such as `["12", "7", "30"]`.

(c) Write a code segment that finds the **smallest** value in `values`, starting from `Integer.MAX_VALUE`. [3]

Exercise 3.4 (c)

3.4 A quiz app stores answers typed by players as `Strings` in `ArrayList<String> typed`, such as `["12", "7", "30"]`.

(c) Write a code segment that finds the **smallest** value in `values`, starting from `Integer.MAX_VALUE`. [3]

(c)

```
int smallest = Integer.MAX_VALUE;
for (int v : values)
{
    if (v < smallest) smallest = v;
}
```

- starts at `Integer.MAX_VALUE`
- traverses `values`, unboxing each
- replaces `smallest` when a value is smaller