1. What index value does the third element of an array have?

The third element of an array has index **2**, because array indexes start at **0**.

---

2. Write the declaration for an array named *quantities* that stores 20 integers.

```
int[] quantities = new int[20];
```

---

3. Write a declaration for an array named *heights* storing the numbers 1.65, 2.15, and 4.95.

```
double[] heights = {1.65, 2.15, 4.95};
```

---

4. Write a for-each statement that displays the integer values stored in an array named *grades*.

```
for (int grade : grades) {
    System.out.println(grade);
}
```

---

6. How does passing an entire array to a method differ from passing a single element of the array?

- **Passing an entire array** passes a **reference** to the whole array.
  → The method can access and possibly **change** the actual array.

- **Passing a single element** passes only the **value** stored at that index (unless it's an object).
  → The method receives just the value, and **changes do not affect** the original array (for primitive types).

---

7. Why are offset array indexes required in some cases?

Offset indexes are used when the program logic starts counting from **1** or another number, but the array still begins at **0**.
They help translate between:

- **human counting** (like “1st item”)
- **computer counting** (index 0)

They also help when using parts of an array or when mapping indexes to real-world values.

---

8. What output is displayed by the statements below?

```
String name = "Elaine";
System.out.println(name.charAt(3));
```

Index positions:
E(0), l(1), a(2), **i(3)**, n(4), e(5)

Output:

```
i
```

---

10. Give an example of when a dynamic array might be a better choice over a regular array.

A dynamic array like an **ArrayList** is better when you **don’t know in advance how many items you will store**.

Example:
Storing user input until the user decides to stop. You don’t know how many items there will be, so an ArrayList can grow automatically.

---

11. How does the ArrayList `indexOf()` method determine equality between the object passed to the method and an element in the array?

`indexOf()` uses the element's **equals()** method to check if two objects are equal.
If `element.equals(target)` is true, it reports the index.

---

12. How can the values of wrapper class objects be compared?

Wrapper class objects (like `Integer`, `Double`, `Character`) should be compared using the **equals()** method rather than `==`.

Example:

```
Integer a = 5;
Integer b = 5;

if (a.equals(b)) {
    System.out.println("Equal!");
}
```