

arrays, the stack, the queue, and the linked list, as ways to hold data



The array, the data structure you already have

*

- an array is already a data structure: a fixed number of slots, all the same type, reached directly by index
- reaching any slot by its index takes the same effort no matter how big the array is, which is what makes an array fast for lookup
- the two structures this chapter adds, the **stack** and the **queue**, are really just RULES for how items go in and come out of a collection, not new storage in themselves
- the syntax for declaring and creating an array is covered in the Programming in Java review, this chapter only needs the idea of indexed slots



`arr[2]` reaches index 2, the value 1, directly, without walking past the slots before it

Three pushes, a pop, and a peek

Q. Starting from an empty stack, run `push(5)`, `push(3)`, `push(8)`, `pop()`, then `peek()`. What does each call return, and what is left on the stack?

- 1 `push(5)`: the stack holds 5, with 5 on top
- 2 `push(3)`: the stack holds 5, 3, with 3 on top
- 3 `push(8)`: the stack holds 5, 3, 8, with 8 on top
- 4 `pop()`: removes and returns 8, the stack is left holding 5, 3, with 3 now on top
- 5 `peek()`: returns 3, the top item, without removing it, the stack still holds 5, 3



the stack after the pop, read left to right as bottom to top, the highlighted cell is the current top

Answer

`pop()` returns 8, `peek()` returns 3, and the stack ends up holding 5 then 3

The stack: last in, first out

- a **stack** only allows adding and removing at one end, called the top, so the last item pushed is always the first item popped
- Java's `java.util.Stack` class provides this directly: `push` adds an item on top, `pop` removes and returns the top item, `peek` returns the top item without removing it
- calling `pop` or `peek` on an empty stack throws an exception, so a real program checks the stack is not empty first
- a stack of function calls is exactly what backs recursion: each call pushes a frame, each return pops one

