

A **stack** has two operations, **push** and **pop**. Pushing adds an element to the **top** of the stack, and **pop** removes the **top** element and returns it.

```
Stack<Integer> s = new ALStack<>();
s.push(4);
s.push(10);
s.push(13);
Integer i = s.pop();
s.push(5);
Integer i2 = s.pop();
```

What number is stored in i?

- A: 4
- B: 10
- C: 13
- D: 5
- E: Something else

What number is stored in i2?

- A: 4
- B: 10
- C: 13
- D: 5
- E: Something else

What is the contents of the stack? (starting at the **top**)

- A. 5, 13, 10, 4
- B. 10, 4
- C. 5, 13
- D. 13, 10, 4
- E. other

A **queue** has two operations, **enqueue** and **dequeue**. Enquing adds an element to the **back** of the queue, and **dequeue** removes the **front** element and returns it.

```
Queue<Integer> q = new ALQueue<>();
q.enqueue(4);
q.enqueue(10);
q.enqueue(13);
Integer i = q.dequeue();
q.enqueue(5);
Integer i2 = q.dequeue();
```

What number is stored in i?

- A: 4
- B: 10
- C: 13
- D: 5
- E: Something else

What number is stored in i2?

- A: 4
- B: 10
- C: 13
- D: 5
- E: Something else

What is the contents of the queue? (starting at the **front**)

- A. 4, 10, 13, 5
- B. 10, 13, 5
- C. 5, 10
- D. 13, 5
- E. other

```
import java.util.ArrayList;

public interface Stack<E> {
    void push(E element);
    E pop();
    int size();
}
// IDEA: Use array lists to implement both
class ALStack<E> implements Stack<E> {
```

}

```
import java.util.ArrayList;

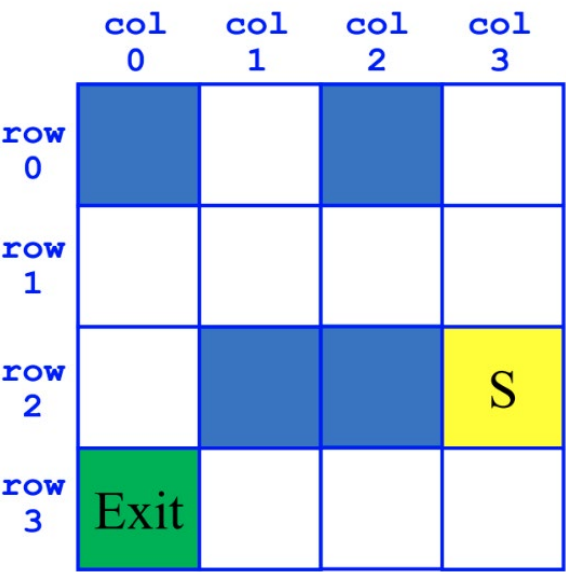
public interface Queue<E> {
    void enqueue(E element);
    E dequeue();
    int size();
}

class ALQueue<E> implements Queue<E> {
```

}

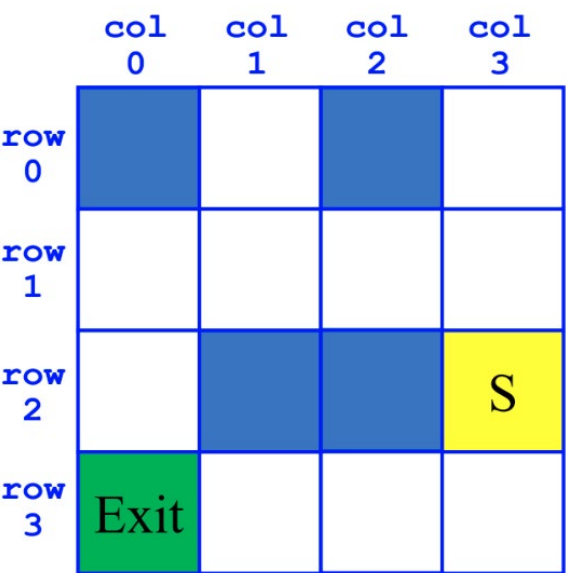
Class ArrayList<E>

void	<u>add</u> (int index, <u>E</u> element)	Inserts the specified element at the specified position.
<u>E</u>	<u>remove</u> (int index)	Removes the element at the specified position in this list.
int	<u>size</u> ()	Returns the number of elements in this list.
<u>E</u>	<u>set</u> (int index, <u>E</u> element)	Replaces the element at the specified position in this list with the specified element.
int	<u>indexOf</u> (<u>Object</u> o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.



SearchForTheExit

Initialize a **Stack** to hold Squares as we search
Mark starting square as visited
Put starting square on task list
While **Stack** is not empty
 Pop square sq from **Stack**
 Mark sq as visited
 If sq is the Exit, we're done!
 For each of square's unseen neighbors (S, W, N, E):
 Set neighbor's previous to sq
 Push neighbor to **Stack**



SearchForTheExit

Initialize a **Queue** to hold Squares as we search
Mark starting square as visited
Put starting square on task list
While **Queue** is not empty
 Dequeue square sq from **Queue**
 Mark sq as visited
 If sq is the Exit, we're done!
 For each of square's unseen neighbors (S, W, N, E):
 Set neighbor's previous to sq
 Enqueue neighbor to **Queue**