```
 // C)Stack minimum
public class MinStack {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> minStack = new Stack<>();

    public void push(int x) {
        stack.push(x);
        if (minStack.isEmpty() || x <= minStack.peek()) {
            minStack.push(x);
        }
    }

    public void pop() {
        if (stack.isEmpty()) {
            return;
        }
        int x = stack.pop();
        if (x == minStack.peek()) {
            minStack.pop();
        }
    }

    public int top() {
        if (stack.isEmpty()) {
            return -1;
        }
        return stack.peek();
    }
```

```
    public int getMin() {
        if (minStack.isEmpty()) {
            return -1;
        }
        return minStack.peek();
    }

    public static void main(String[] args) {
        MinStack stack = new MinStack();
        stack.push(3);
        stack.push(1);
        stack.push(2);
        stack.push(5);
        stack.push(0);

        System.out.println("Minimum element: " + stack.getMin()); // 0
        stack.pop();
        stack.pop();
        System.out.println("Minimum element: " + stack.getMin()); // 1
        stack.push(-1);
        System.out.println("Minimum element: " + stack.getMin()); // -1
    }
}
```

To implement a Stack with a min function that returns the minimum element in the stack in constant time (O(1)), we can use two stacks: a main stack to store the elements and an auxiliary stack to store the minimums.

Each time an element is pushed onto the main stack, we compare it with the top element of the auxiliary stack. If it is smaller or equal, we push it onto the auxiliary stack as well. When an element is

popped from the main stack, we check if it is the same as the top element of the auxiliary stack. If it is, we also pop it from the auxiliary stack. The min function simply returns the top element of the auxiliary stack.

Here's an implementation of the Stack with a min function in Java:

Bonus 1: A real-world use case where a stack is a better data structure than an array is in the implementation of a web browser's back button functionality. When the user visits a website, the browser can push the URL onto a stack. When the user clicks the back button, the browser can simply pop the top URL from the stack to return to the previous page. This approach is more efficient than storing all visited URLs in an array and iterating through it to go back to a previous page. Additionally, the stack approach takes up less memory than an array since it only stores the most recent URLs