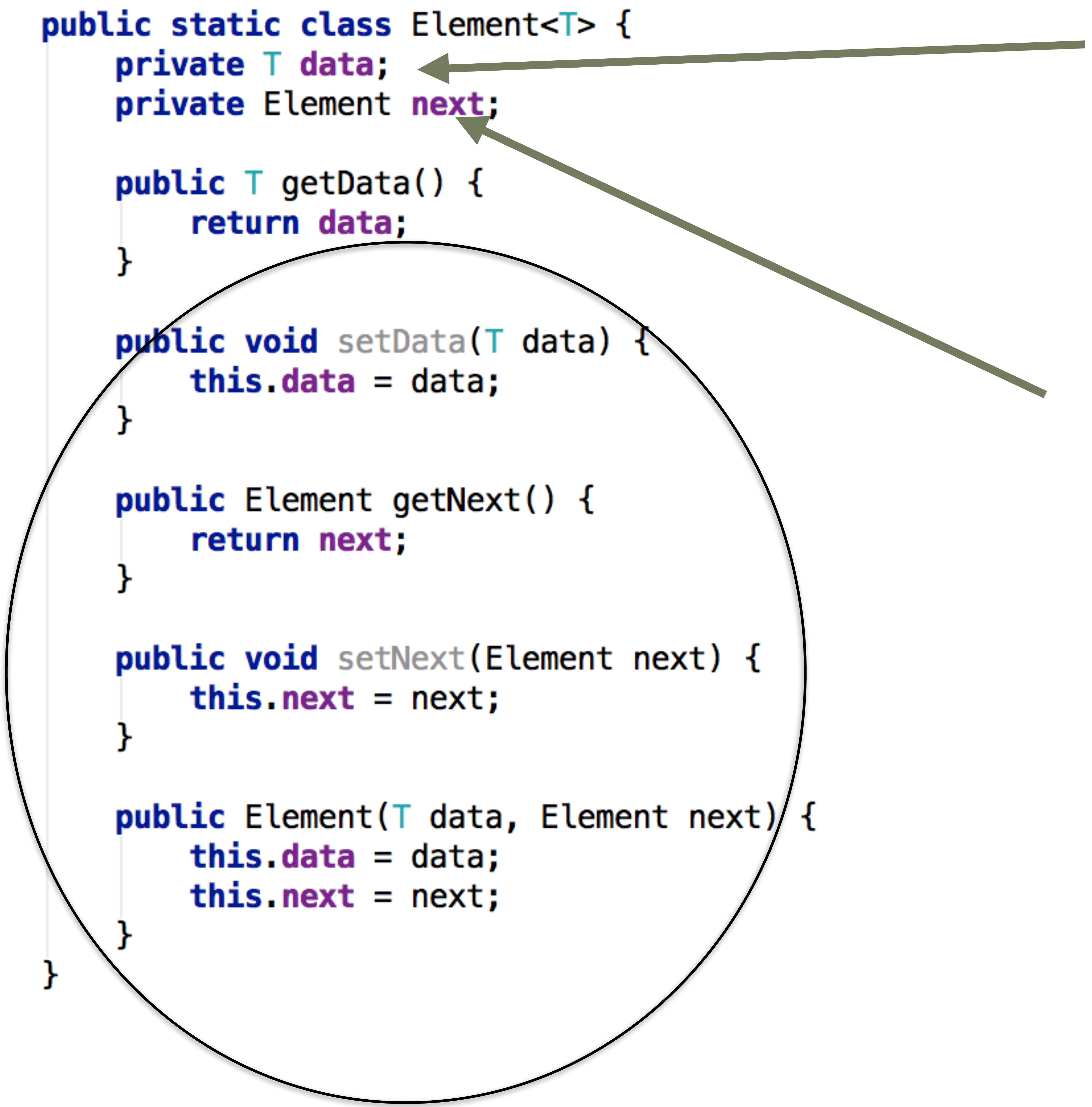
LET'S BUILD A STACK

A SINGLE ELEMENT IN THE LINKED LIST



```
public static class Element<T> {
    private T data;
    private Element next;

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }

    public Element getNext() {
        return next;
    }

    public void setNext(Element next) {
        this.next = next;
    }

    public Element(T data, Element next) {
        this.data = data;
        this.next = next;
    }
}
```

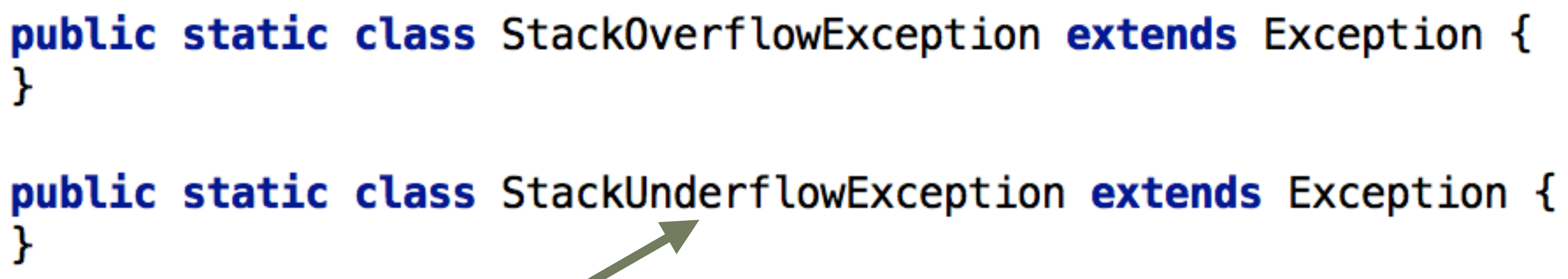
A GENERIC LINKED LIST ELEMENT WHICH CAN STORE DATA OF ANY TYPE

A NEXT POINTER WHICH POINTS TO THE NEXT ELEMENT IN THE LIST

HELPER METHODS

EXCEPTIONS THROWN

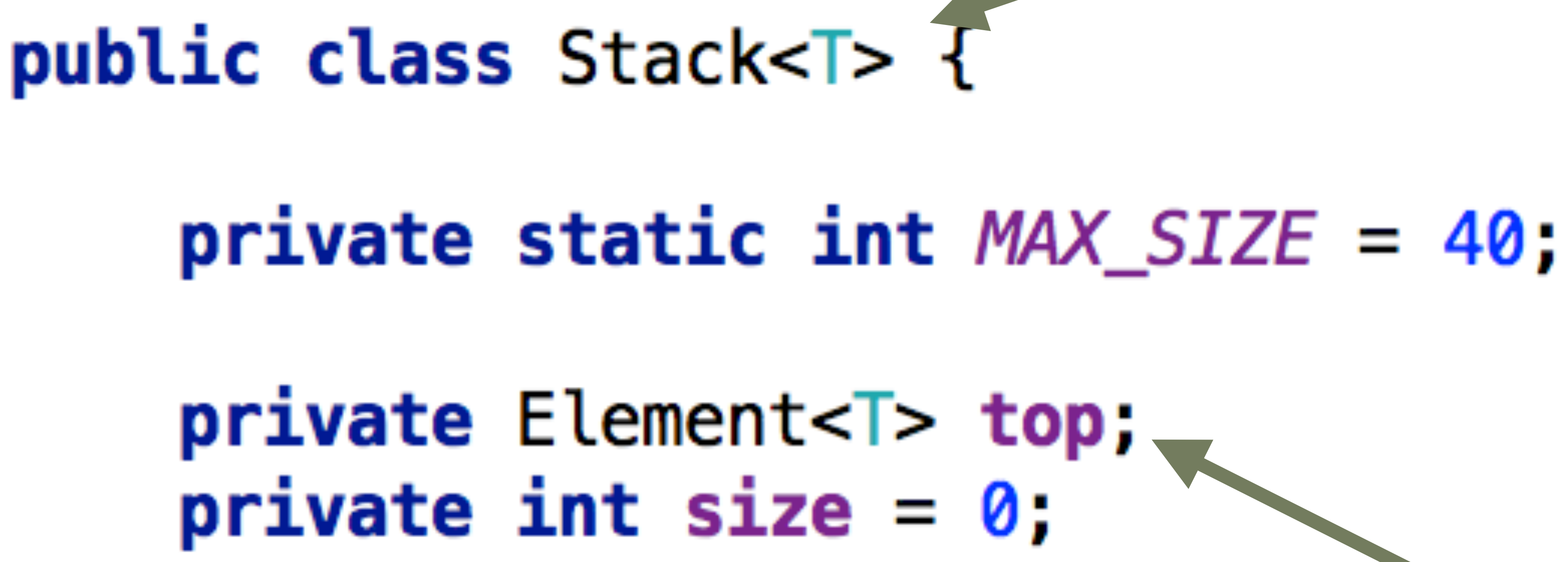
PUSHING INTO A FULL STACK



```
public static class StackOverflowException extends Exception {
}

public static class StackUnderflowException extends Exception {
}
```

POPPING FROM OR PEEKING INTO
AN EMPTY STACK

GENERIC STACK CLASS - CODE SNIPPET



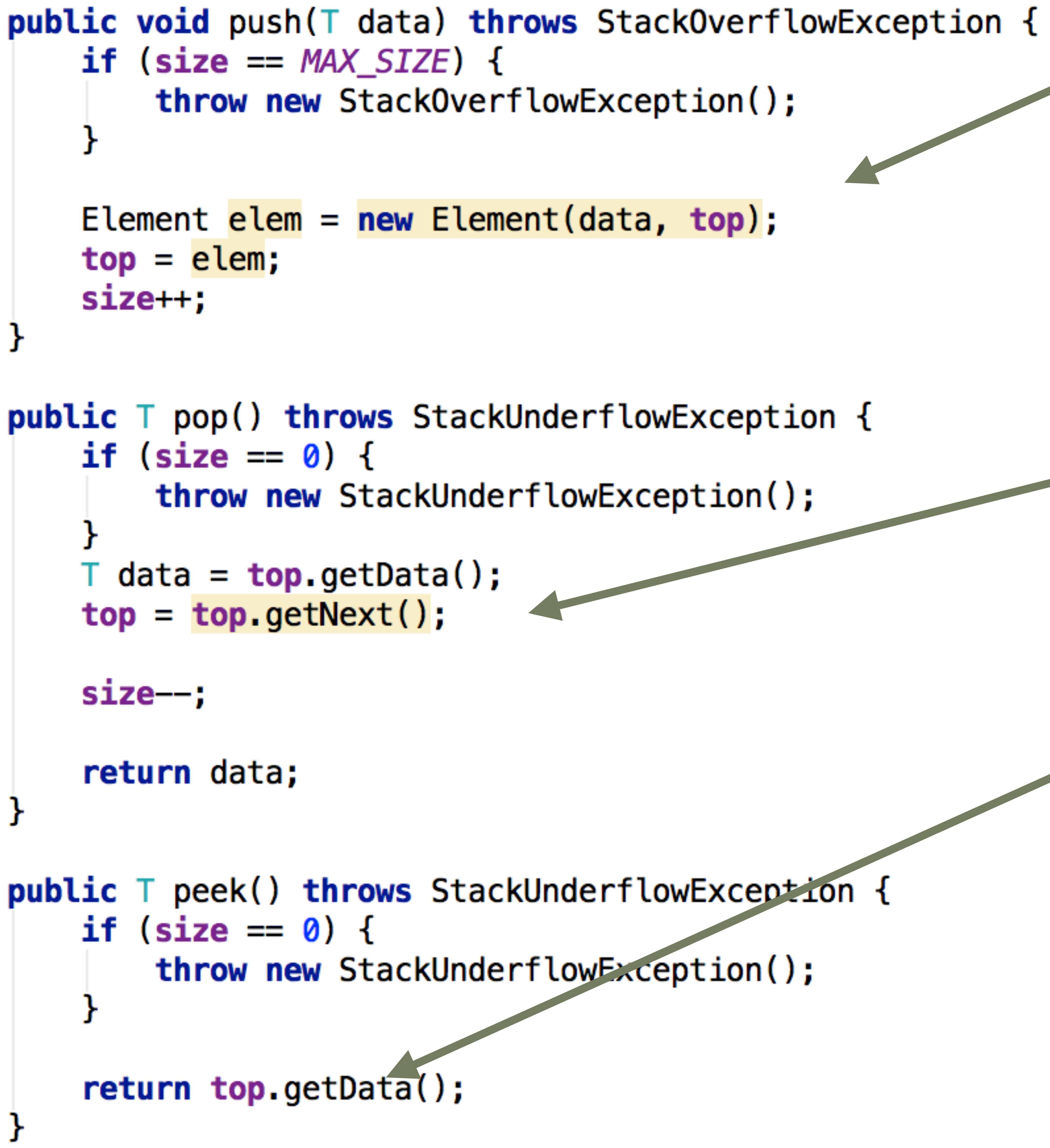
```
public class Stack<T> {

    private static int MAX_SIZE = 40;

    private Element<T> top;
    private int size = 0;
```

A GENERIC CLASS CAN BE INSTANTIATED WITH ANY DATA TYPE - WHICH MEANS THE STACK CAN HOLD ANY DATA TYPE

POINTS TO THE TOPMOST ELEMENT IN THE STACK, THE ONE WHICH CAN BE POPPED OR PEEKED AT

TRACK THE SIZE OF THE STACK AT EVERY PUSH, POP. THIS MEANS THE STACK SIZE OPERATION CAN BE CONSTANT TIME

PUSH, POP, PEEK - CODE SNIPPET



```java
public void push(T data) throws StackOverflowException {
    if (size == MAX_SIZE) {
        throw new StackOverflowException();
    }

    Element elem = new Element(data, top);
    top = elem;
    size++;
}

public T pop() throws StackUnderflowException {
    if (size == 0) {
        throw new StackUnderflowException();
    }
    T data = top.getData();
    top = top.getNext();

    size--;

    return data;
}

public T peek() throws StackUnderflowException {
    if (size == 0) {
        throw new StackUnderflowException();
    }

    return top.getData();
}
```

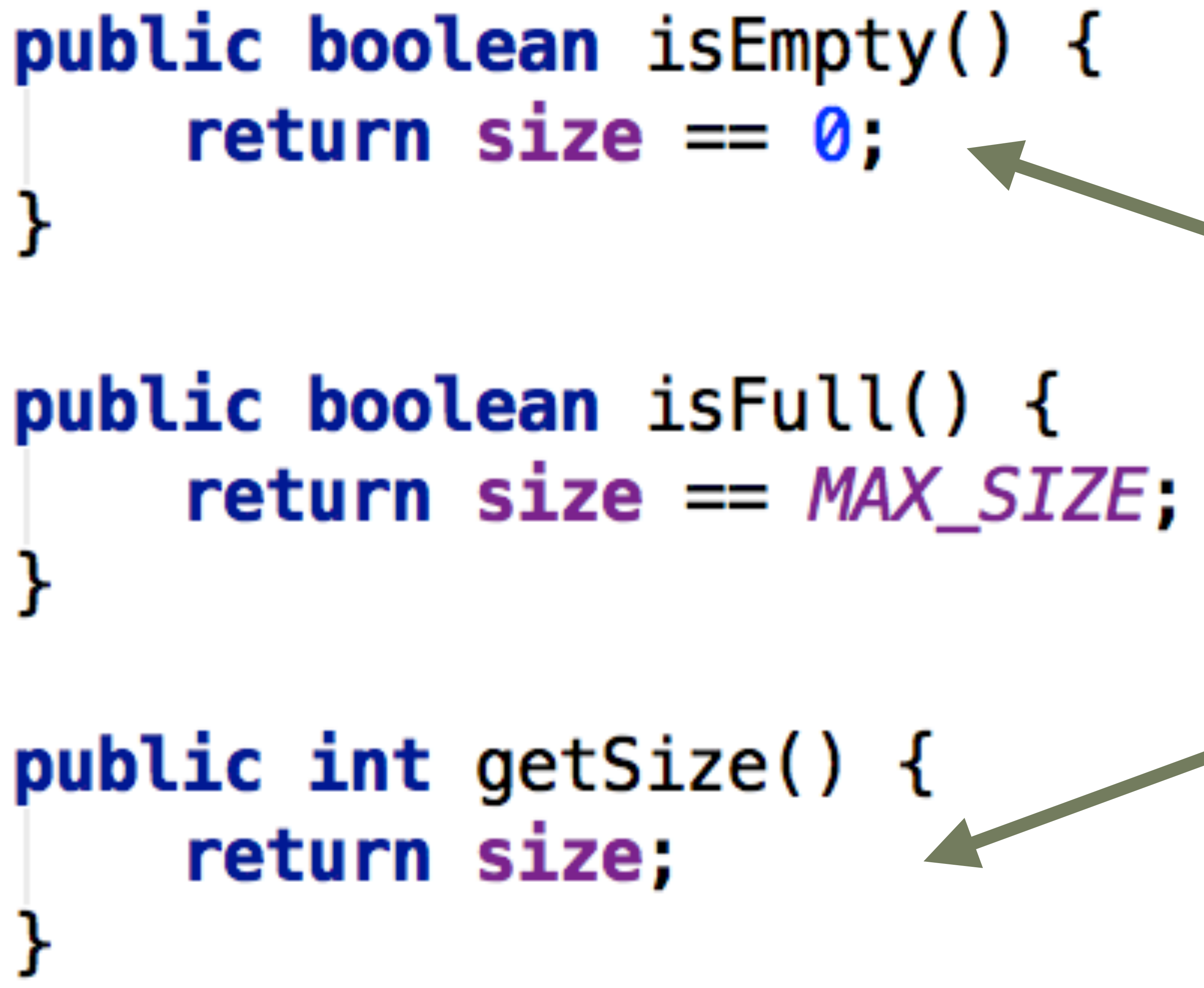
CREATE A NEW ELEMENT TO HOLD THE DATA AND POINT TOP TO IT. MAKE SURE YOU INCREMENT THE SIZE OF THE STACK

REMOVE THE TOP ELEMENT FROM THE STACK AND RETURN IT'S DATA

JUST RETURN THE DATA OF THE TOP ELEMENT, DO NOT REMOVE IT

NOTE THE EXCEPTIONS THROWN BY EACH OF THESE METHODS

ISEMPTY, ISFULL, SIZE - CODE SNIPPET



```java
public boolean isEmpty() {
    return size == 0;
}

public boolean isFull() {
    return size == MAX_SIZE;
}

public int getSize() {
    return size;
}
```

THESE METHODS BECOME SUPER SIMPLE WITH THE SIZE VARIABLE

THE STACK - PERFORMANCE AND COMPLEXITY

PUSH AND POP FROM A STACK IMPLEMENTED IN THIS WAY IS O(1), CONSTANT TIME COMPLEXITY

IS EMPTY AND IS FULL IS ALSO O(1)

THE USE OF THE "SIZE" VARIABLE MAKES GETTING THE SIZE OF THE STACK ALSO O(1)

SPACE COMPLEXITY IS O(N)

WHERE CAN STACKS BE USED?

IMPLEMENTING UNDO IN AN APPLICATION

IMPLEMENTING THE BACK BUTTON ON THE WEB BROWSER

HOLDING THE MEMORY FOR RECURSIVE CALLS IN A PROGRAMMING LANGUAGE

TRANSLATING INFIX NOTATION FOR EXPRESSIONS TO POSTFIX