CMPT 225: Data Structures & Programming – Unit 08 – Stacks

Dr. Jack Thomas
Simon Fraser University
Spring 2021

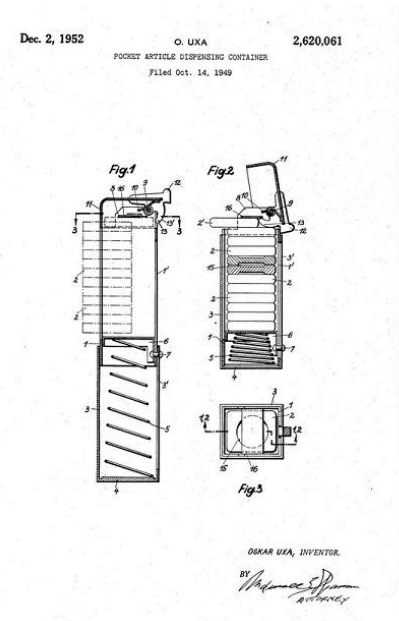
Today's Topics

- The Stack!
- ADT for the Stack
- The Stack in Java
- Implementing and Analyzing

What is a Stack?

- Exactly what it sounds like – a stack, a set of things piled up on each other.
- **A Stack is a data structure**, with an ADT and implementations in most languages.
- It follows the “First In, Last Out” rule, or **FILO**.
- Each element added to the Stack buries the previous element.
- When you go to start taking things off the stack, you have to **start with the most recent element**, then the next-most recent, and so on until you **get down to the first element**.

Think of a PEZ Dispenser



- Wait has anyone here even used a PEZ dispenser?
- There's got to be a less embarrassingly old-timey example.

Image credit: https://www.smithsonianmag.com/innovation/how-pez-evolved-from-anti-smoking-tool-to-beloved-collectors-item-180976545/

Stack: The ADT

- A Stack stores a set of objects.
- Follows FILO (first-in-last-out).
- Standard Stack operations include:
  - **Push**: Add an element to the top of the Stack.
  - **Pop**: Remove the top element.
  - **Top**: Return what's on top of the stack without removing it.
  - **Size**: How many things are on the Stack?
  - **Empty**: Is the stack empty? Yes or no.

Stack Examples In Software

- Your **web browser back button**, which pushes each page you visit onto the stack, then pops them back off again from latest to oldest.
- Most **undo functions** work the same way, saving a record of each action taken so that the latest action is always on top.
- Stacks are useful in general for tracking a **linear history of events** you might have to start moving backward through.

Stack in Java

- Java has a standard Stack in java.util, which takes in Java objects and includes methods like push(), pop(), peek() (like top but a funnier name), size(), and empty().

```
import java.util.Stack;

public class Main {

    public static void main(String[] args)
    {
        Stack<Integer> exampleStack = new Stack<Integer>();
        exampleStack.push( item: 1);
        exampleStack.push( item: 2);
        exampleStack.push( item: 3);
        System.out.println(exampleStack.pop());
        System.out.println(exampleStack.pop());
        System.out.println(exampleStack.pop());
    }
}
```

```
3
2
1
```

Stacks, Arrays, and Lists

- A Stack is one level **more complex than an array or list**. It defines how the data within the structure is accessed, but not how it's stored.
- For that storage, do Stacks use an array, or a list?
- **It can use either**. You can build a Stack using an array or a list as the basis, so long as the methods like push() and pop() work as expected.
- The Java Stack is actually based on a Vector, an old legacy class that works like a growable array.

Implementing an Array-Based Stack

- Let's implement our own version of Stack in Java for storing integers.

```
class ArrayStack<Integer> {
    protected int capacity;
    public static final int DEFAULTCAPACITY = 1000;
    protected int elements[];
    protected int top = -1;
    public ArrayStack() {
        this(DEFAULTCAPACITY);
    }
    public ArrayStack(int cap) {
        capacity = cap;
        elements = new int[capacity];
    }
```

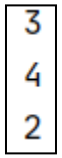
```
public int size() {
    return (top + 1);
}
public boolean isEmpty() {
    return (top < 0);
}
```

Implementing an Array-Based Stack

```
public void push(int element) throws RuntimeException {
    if (size() == capacity)
        throw new RuntimeException("Stack is full.");
    elements[++top] = element;
}
public int top() throws RuntimeException {
    if (isEmpty())
        throw new RuntimeException("Stack is empty");
    return elements[top];
}
public int pop() throws RuntimeException {
    int element;
    if (isEmpty())
        throw new RuntimeException("Stack is empty.");
    element = elements[top];
    top--;
    //for non-primitives, elements[top--] = null;
    return element;
}
```

Implementing an Array-Based Stack

```
ArrayStack<Integer> testStack = new ArrayStack<Integer>();
testStack.push( element: 2);
testStack.push( element: 4);
testStack.push( element: 3);
System.out.println(testStack.pop());
System.out.println(testStack.pop());
System.out.println(testStack.pop());
```



Analyzing Our Stack

- Time analysis per method:
  - **size:** O(1)
  - **isEmpty:** O(1)
  - **top:** O(1)
  - **push:** O(1)
  - **pop:** O(1)
- The only drawback: array requires a fixed size on creation, meaning it's either full or wasting memory.
- We could try implementing with a list instead!

Recap – You've Reached The First Slide I Made!

- **Stacks** are a type **of data structure** that follows the **first-in-last-out** rule for storing objects.
- Useful for **tracking things in reverse order**, like a history of events.
- **Java** has a **built-in Stack class** you can use.
- You can also define your own Stack, which can be based on **an array or a list** (or a vector, if you're old-school).
- Try making your own Stack at home by just leaving stuff in piles on the floor!