Assignment 3: Stacks

Part 1: Stacks

CS3305/W01 Data Structures

Casey Hampson

September 3, 2024

Program Output

Pushing

```
1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:1
1) Stack A
2) Stack B
Please choose an option:1
Enter a value to push: 1

1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:1
1) Stack A
2) Stack B
Please choose an option:2
Enter a value to push: 2

1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:1
1) Stack A
2) Stack B
Please choose an option:1
Enter a value to push: 3
```

Popping

```
1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:2
1) Stack A
2) Stack B
Please choose an option:1
The popped value is: 3

1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:2
1) Stack A
2) Stack B
Please choose an option:1
The popped value is: 1

1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:2
1) Stack A
2) Stack B
Please choose an option:2
The popped value is: 2
```

Pushing/Popping into Full/Empty Stack

I only show the last successful push/pop in these screenshots; since this works for any size stack, I will choose not to clutter the screen.

```
1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:1
1) Stack A
2) Stack B
3) Cancel/Return to main menu
Please choose an option:2
Enter a value to push: 2

1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:1
Cannot push, the stack is full!

1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:
```

```
1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:2
1) Stack A
2) Stack B
3) Cancel/Return to main menu
Please choose an option:2
The popped value is: 2

1) Push an integer onto a stack
2) Pop an integer from a stack
3) Quit
Please choose an option:2
1) Stack A
2) Stack B
3) Cancel/Return to main menu
Please choose an option:2
Cannot pop from Stack B, it's empty!
1) Stack A
2) Stack B
3) Cancel/Return to main menu
Please choose an option:
```

Source Code

```java
// Name: Casey Hampson
// Class: CS 3305/W01
// Term: Fall 2024
// Instructor: Sharon Perry
// Assignment: 3-Part-1-Stacks

import java.util.Scanner;

class StackPair {
    final private int[] Stack; // the main stack array
    // three variables for determining sizes/lengths
    final private int stack_size;
    private int len_a;
    private int len_b;

    // initialize everything
    StackPair(int capacity) {
        this.Stack = new int[capacity];
        this.stack_size = capacity;
        this.len_a = 0;
        this.len_b = 0;
    }

    // checks if the sum of the lengths of the two stacks are equal to the capacity of the stack;
    // if so, then it must be full
    boolean IsFull()  { return (this.len_a + this.len_b == this.stack_size); }

    // we need to separately check the emptiness, since we could have something
    // in stack A but not stack B, but if we try to pop from stack B, it wouldn't work
    // despite it not being ``globally'' empty
    boolean IsStackAEmpty() { return (this.len_a == 0); }
    boolean IsStackBEmpty() { return (this.len_b == 0);}

    /* These following methods I utilize the increment operator to keep everything in one line */
    /* Additionally, a -1 is needed in the StackA methods to deal with 0-based indexing  */

    // pushes x into stack A
    void    PushA(int x)    { this.Stack[this.len_a++] = x; }
    // pushes x into stack B
    void    PushB(int x)    { this.Stack[(this.stack_size - 1) - this.len_b++] = x; }
    // pops the top value from stack A
    int     PopA()          { return this.Stack[this.len_a-- - 1]; }
    // pops the top value from stack A
    int     PopB()          { return this.Stack[this.stack_size - this.len_b--];}
}

public class P1 {
    public static Scanner input_scanner;
    final static int STACK_SIZE = 1; // small initial stack capacity for simplicity

    // enumerated list of possible menu choices for better readability
    static enum MENU_CHOICE {
```

```
    PUSH,
    POP,
    STACKA,
    STACKB,
    CANCEL,
    QUIT,
    ERROR
};

// grab the user's choice of either pushing or popping
static MENU_CHOICE MainMenu() {
    System.out.printf("""

        1) Push an integer onto a stack
        2) Pop an integer from a stack
        3) Quit
        Please choose an option: """
    );
    int choice = Integer.parseInt(input_scanner.nextLine());
    switch (choice) {
        case 1  -> {return MENU_CHOICE.PUSH;}
        case 2  -> {return MENU_CHOICE.POP;}
        case 3  -> {return MENU_CHOICE.QUIT;}
        default -> {return MENU_CHOICE.ERROR;}
    }
}

// grab the users choice for which stack to push/pop to
static MENU_CHOICE StackMenu() {
    System.out.printf("""
        1) Stack A
        2) Stack B
        3) Cancel/Return to main menu
        Please choose an option: """
    );
    int choice = Integer.parseInt(input_scanner.nextLine());
    switch (choice) {
        case 1 ->  {return MENU_CHOICE.STACKA;}
        case 2 ->  {return MENU_CHOICE.STACKB;}
        case 3 ->  {return MENU_CHOICE.CANCEL;}
        default -> {return MENU_CHOICE.ERROR;}
    }
}


public static void main(String[] args) {
    // instantiate the scanner and stack pair objects
    input_scanner = new Scanner(System.in);
    StackPair stack_pair = new StackPair(STACK_SIZE);

    // grab the main menu choice, ensure it is a valid option
    MENU_CHOICE main_choice = MainMenu();
    while (main_choice.equals(MENU_CHOICE.ERROR)) {
        System.out.printf("Invalid option!\n");
```

```
        main_choice = MainMenu();
    }

    while (!main_choice.equals(MENU_CHOICE.QUIT)) {
        // if we are pushing, make sure the stack isn't full, empty is slightly different
        if (stack_pair.IsFull() && main_choice.equals(MENU_CHOICE.PUSH)) {
            System.out.printf("Cannot push, the stack is full!\n");
            main_choice = MainMenu();
            continue;
        }


        // grab which stack to use
        MENU_CHOICE stack_choice = StackMenu();
        while (stack_choice.equals(MENU_CHOICE.ERROR)) {
            System.out.printf("Please enter a valid choice.\n");
            stack_choice = StackMenu();
        }
        if (stack_choice.equals(MENU_CHOICE.CANCEL)) break;

        // now we push or pop from whichever stack was chosen
        if (main_choice.equals(MENU_CHOICE.PUSH)) {
            System.out.printf("Enter a value to push: ");
            int val = Integer.parseInt(input_scanner.nextLine());

            switch (stack_choice) {
                case MENU_CHOICE.STACKA -> {stack_pair.PushA(val);}
                case MENU_CHOICE.STACKB -> {stack_pair.PushB(val);}
                default -> {} // unreachable
            }
        } else if (main_choice.equals(MENU_CHOICE.POP)) {
            switch (stack_choice) {
                case MENU_CHOICE.STACKA -> {
                    if (stack_pair.IsStackAEmpty()) {
                        System.out.printf("Cannot pop from Stack A, it's empty!\n");
                        continue;
                    }
                    System.out.printf("The popped value is: %d\n", stack_pair.PopA());
                }
                case MENU_CHOICE.STACKB -> {
                    if (stack_pair.IsStackBEmpty()) {
                        System.out.printf("Cannot pop from Stack B, it's empty!\n");
                        continue;
                    }
                    System.out.printf("The popped value is: %d\n", stack_pair.PopB());
                }
                default -> {} // unreachable
            }
        }


        // ensure we keep grabbing the main choice to keep running the program, if desired
        main_choice = MainMenu();
    }
```

```
        // close resources
        input_scanner.close();
    }
}
```