MACQUARIE University

Faculty of Science and Engineering

COMP125 Fundamentals of Computer Science
Workshop Week 12

Learning outcomes

This week we look at stacks. By the end of this session, you should

a. have a better understanding of how stacks operate;

b. be able to use stacks to solve simple problems.

Import the java project from the archive file `stacks_workshop_start.zip`

Questions

1. a. Using pen and paper, add strings "Are", "you", "my", "mummy?" to a stack in the given order. What message would be displayed when removing elements from the stack?

   b. Implement a) in Java. Namely, create a stack of `String`, add items "Are", "you", "my", "mummy?" to the stack and use a loop to display all the elements in the stack, from top to bottom, separated by a space.

Solution:

```
Stack<String> stk = new Stack<String>();
Stack<String> stk2 = new Stack<String>();
stk.push("Are");
stk.push("you");
stk.push("my");
stk.push("mummy?");
while(!stk.isEmpty()) {
        System.out.print(stk.peek()+"␣");
        stk2.push(stk.pop());
}
while(!stk2.isEmpty()) {
        stk.push(stk2.pop());
}
System.out.println();
```

2. a. How would you display the elements of a stack in the order they are entered in the stack, i.e. from bottom to top?

   b. The class `Card` deals with playing cards. It has methods to set and display the value of a card. In the class `Hand`, write a method `makeHand` that takes an integer `n` and returns a stack of `n` random cards. The method should also display the values of the card as they are generated and added to the stack.

   c. Write a method `reverseDisplay` that displays the cards in the order they are entered in the stack.

Solution: We can use another stack or adopt a recursive approach to display the items stored in a stack in the order they were entered.

To write `makeHand`, we make use of the methods `setRandom` and `display`. The cards are added to the stack using `push`.

See `stacks_workshop_solution.zip` for an implementation

```
public static Stack<Card> makeHand(int n) {
    Stack<Card> stack = new Stack<Card>();
    for(int i = 1; i <= n; i++) {
        Card card = new Card();
        card.setRandom();
        System.out.println(card);
        stack.push(card);
    }
    System.out.println();
    return stack;
}

public static void reverseDisplay(Stack<Card> s) {
    Stack<Card> s2 = new Stack<Card>();
    while(!s.empty()) {
        s2.add(s.pop());
    }
    while(!s2.empty()) {
        s2.pop().display();
    }

}

public static void reverseRecDisplay(Stack<Card> s) {
    if(!s.empty()) {
        Card card = s.pop();  //the card is displayed after the call
                              //but has been saved before
        reverseRecDisplay(s);
        card.display();
    }
}
```

3. **(assessed)** In class `StackDemo`, complete the method `countEven` that when passed a stack of integers, returns the number of even numbers. The stack must remain unchanged after the method finishes execution. Return `null` if the stack is empty.

Solution:

```
        public static int countEven(Stack<Integer> stk) {
                if(stk == null)
                        return 0;

                int result = 0;
                for(Integer item: stk){
                        if(item % 2 == 0) {
                                result++;
                        }
                }
                return result;
        }
```

4. **(assessed)** In class `StackDemo`, complete the method `alternateItems` that when passed a stack of integers, returns a stack containing every alternate item, starting from the top item, from the passed stack. For example, if the stack passed is [6, 3, 1, 8] (where 8 is the top item), returns the stack [3, 8], and when the stack passed is [2, 6, 3, 1, 8], returns the stack [2, 3, 8].

Solution:

```
public static Stack<Integer> alternateItems(Stack<Integer> stk) {
        if(stk == null)
                return null;

        Stack<Integer> temp = new Stack<Integer>();

        while(!stk.isEmpty()) {
                temp.push(stk.pop());
                if(!stk.isEmpty())
                        stk.pop();
        }

        Stack<Integer> result = new Stack<Integer>();

        while(temp.isEmpty() == false) {
                result.push(temp.pop());
        }

        return result;
        }
```

5. **(assessed)** Write a method that when passed an array of integers, returns the array reversed, using a stack.

Solution:

```
public static int[] reverse(int[] a) {
        Stack<Integer> stk = new Stack<Integer>();
        for(int item: a)
                stk.push(item);
        int[] result = new int[a];
        int i = 0;
        while(!stk.isEmpty()) {
                result[i++] = stk.pop();
        }
        return result;
}
```

Revision questions

6. Write a method that when passed an array of integers, returns the sum of all items
7. Write a method that when passed an array of integers, returns the average of all items
8. Write a method that when passed an array of `double` values, returns `true` if it is sorted in ascending order (such that each item is more than or equal to the previous item), and `false` otherwise.
9. Write a method that when passed two arrays of `double` values, returns `true` if they are exactly the same - item for item, and `false` otherwise.
10. Write a method that when passed two arrays of `double` values, returns `true` if they are mutually reverse, and `false` otherwise.
11. Write a method that when passed an array of `double` values, returns `true` if all items are positive, and `false` otherwise.
12. Write a method that when passed an integer array, returns `true` if all items are even, and `false` otherwise.
13. Write a method that when passed an array of `double` values, returns `true` if at least one item is positive, and `false` otherwise.
14. Write a method that when passed an integer array, returns `true` if at least one item is even, and `false` otherwise.
15. Write a method that when passed an integer array, returns `true` if the array follows a fibonacci sequence, that is each item of the array is the sum of the previous two items (with the first two items acting as the *seed*), and `false` otherwise.
16. Write a method that when passed an array of `double` values, returns `true` if no two items of the array are the same, and `false` otherwise.
17. Perform all questions from 6 to 16, using the following data structures instead of arrays,
    - linked list
    - array list
    - stack
18. What is the value of `result` when the following java class is executed?

```
1  public class Rec {
2          public static int foo(int n) {
3                  if(n == 0)
4                          return 0;
5                  return n%10 + foo(n/10);
6          }
7
8          public static void main(String[] args) {
9                  int result = foo(32768);
10         }
11 }
```

19. Write a recursive method, that when passed an integer n (assume $n \geq 0$), computes and returns the sum of the first n positive integers.
20. Write a recursive method, that when passed an integer n (assume $n \geq 0$), computes and returns the product of the first n positive integers.
21. Write a recursive method, that when passed a double x an integer n (assume $n \geq 0$), computes and returns x^n.

22. Write definition for the following classes including data members or instance variables (limit them to a maximum of 3) with appropriate data types, getters, setters with appropriate validation, default constructor, one or more paramterized constructors, and the following instance methods - equals(XYZ obj):boolean, compareTo(XYZ obj):int (where obj is an object of the same class XYZ as the calling object), toString():String.

    a. Circle
    b. Rectangle
    c. Square
    d. Person
    e. Cube
    f. Sphere

23. **(Pretty advanced)** Write a method that returns a 9 x 9 valid sudoku puzzle (that has at least one solution), partially filled. The design of your code should be such that the puzzle numbers and the partial filling must be *clever*.