GIFT School of Engineering and Applied Sciences

Fall 2022

CS-242: Data Structures and Algorithms

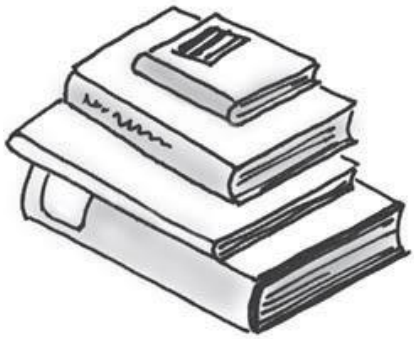
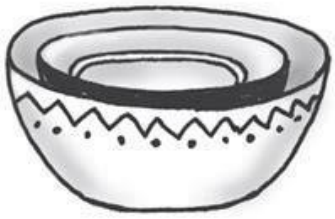
Week-6 Manual

Stack

v1.0
6/11/2019

Stack

In everyday life, a stack is a familiar thing. You might see a stack of books on your desk, a Stack of dishes in the cafeteria, a stack of towels in the linen closet, or a stack of boxes in the attic. When you add an item to a stack, you place it on top of the stack. When you remove an item, you take the topmost one. This topmost item is the last one that was added to the stack.
So when you remove an item, you remove the item added most recently. That is, the last item added to the stack is the first one removed.
In spite of our examples of a stack, everyday life usually does not follow this last-in, first-out, or LIFO, behavior. Although the employee hired most recently is often the first one fired during a layoff, we live in a first-come, first-served society. In the computer science world, however, last-in, first-out is exactly the behavior required by many important algorithms. These algorithms often use the abstract data type stack, which is an ADT that exhibits a last-in, first-out behavior. For example, a compiler uses a stack to interpret the meaning of an algebraic expression, and a run-time environment uses a stack when executing a recursive method.


Task #1: Wrapper Stack

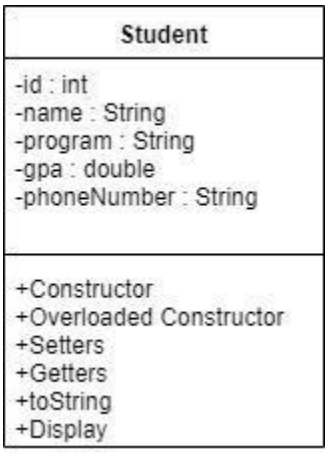
1. Read **`StackInterface.java, LinkedStack.java`**, **`ArrayStack.java`** and **`Driver.java`** files carefully.

Note: You may copy these files from the **`Code`** folder.

2. Create another class **`WrapperStack.java`** with the main method.
3. Write a Display method, which display all the entries from the stack by using the **`pop`** method. You may use the following header for display method **`displayStack(StackInterface stack)`**
4. Create a new stack **`IntegerStack`** using the **`LinkedStack`** class of type **Integer**.
5. Push 3 new entries in stack.
6. Display the stack using the **`displayStack`** method.
7. Repeat the Same process from point 4 to onwards for the types **Double** and **Character** as well.

Task #2: Student Stack

1. Complete the implementation the class given below



2. Create another class **`StudentStack.java`** with the main method.
3. Write another method to display the stack.
8. **Note:** You may use the **`displayStack(StackInterface stack)`** method from the **`WrapperStack.java`** you have already implemented in **Task-1**
4. Create a new stack **`studentStack`** using the **`LinkedStack`** class.
5. Push 2 new objects of student in the **`studentStack`** you have created.
6. Display the list using the **`displayStack`** method.

Task #3: Check for Balanced Delimiters

1. Write a class **`Expressions.java`** with main method as well.
2. Write a method in **`Expressions.java`** which Check for Balanced Delimiters in an Infix Expression, if it is balanced the method returns true and false otherwise. You may use the following header **`public Boolean CheckForBalanacedDelimiters(String expression)`**
    a. **`[{(a+b) + (a-2 + (a-2))}]`** is a valid expression.
    b. **`[{a+(a+b + (a)}]`** is an invalid expression.
    c. **`](a+2)`** is an invalid expression.
    d. **`[{(a+b)}]`** is a valid expression.
3. Call this method in main method for two Expressions, one for which this method will return true and one for which this method will return false.

Task #4: Infix to Postfix Conversion

1. Add another method in **`Expressions.java`** in such a way it converts the infix expression to postfix expression and return it, You may use the following header Public **`String InfixToPostfix(String expression).`**
   Note: This method takes an expression of infix and convert it into a postfix of string and then return the String of Postfix.
2. Call this method in main method in Expressions for some infix expression and convert it into postfix expression.

Task #5: Postfix evaluation

1. Add another method in **`Expressions.java`** which evaluates the Postfix expression, you may use the following header **`public int EvaluatePostFix(String expression).`**
   Note: This method takes an expression of postfix and evaluate it by using stack and return the integer result.
2. Call this method in main method of Expressions for some postfix expression.