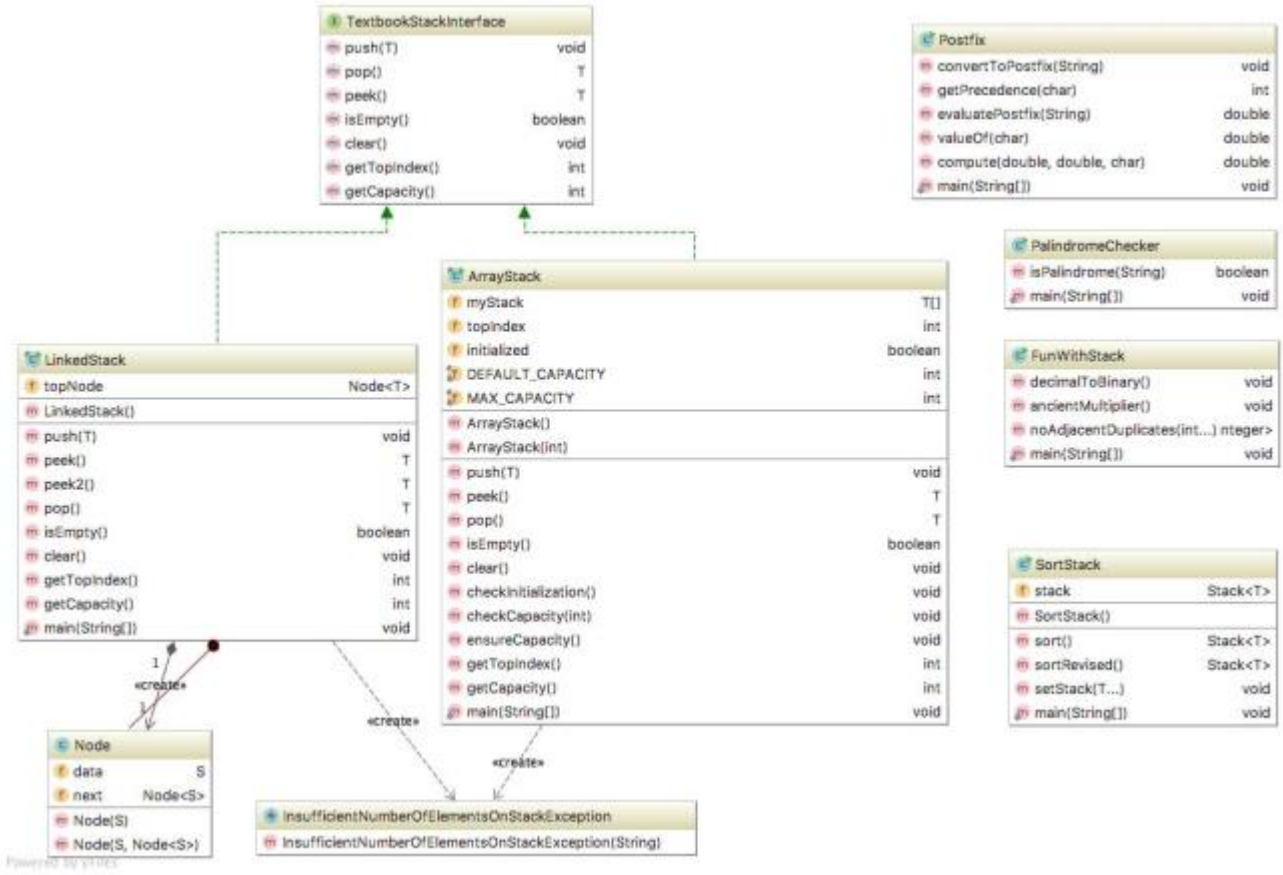
Comp151 Lab04

You are to write **six** independent applications that utilize stack as per descriptions below:

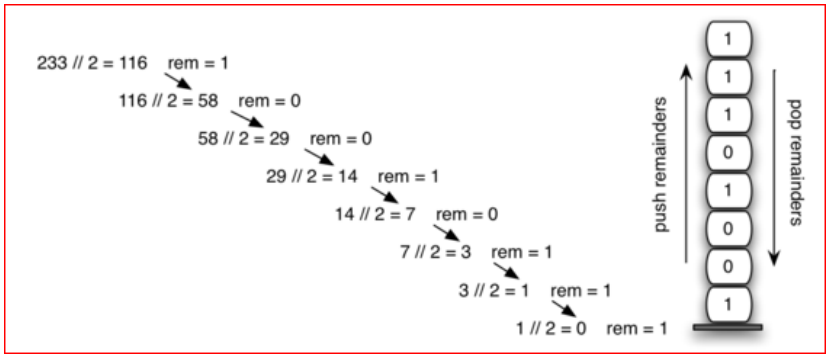
UML DIAGRAM



Application #1

Open `FunWithStack.java`, it contains three methods defined there as skeletons, and main to test them. Utilize `Stack` class defined in `java.util`, create the stacks as follow: `Stack<Integer> myStackName = new Stack<>();`

1. "Decimal to Binary Conversion":
    a. utilizing a stack trace the algorithm on paper
    b. analyze the sample run
    c. describe the algorithm in pseudocode
    d. implement and test



2. "Ancient Egyptian Multiplication":
    a. go to http://en.wikipedia.org/wiki/Ancient_Egyptian_multiplication and analyze the given algorithm
    b. analyze sample runs
    c. why is this method significant to computer scientists?
    d. utilizing two stacks trace the algorithm on paper
    e. describe the algorithm in *pseudocode*
    f. implement and test
3. "Elimination of Adjacent Duplicates":
    a. given an array of integers create `ArrayList<Integer>` that contains all the numbers remaining after eliminating all adjacent duplicates in one pass using one stack
    b. analyze sample runs
    c. describe the algorithm in pseudocode
    d. implement and test

SAMPLE RUN:

```
*** DECIMAL TO BINARY CONVERTER ***

Please enter a positive integer, or type "stop"
6
6 in binary is --> 110

Please enter a positive integer, or type "stop"
12345
```

```
12345 in binary is --> 11000000111001

Please enter a positive integer, or type "stop"
stop
Done with conversion.

*** ANCIENT MULTIPLIER ***

Please enter operand1, or type "stop"
12345
Please enter operand2
6789
The smaller operand is: 6789; and the larger operand is: 12345
--> Creating the mapping table:
1 --> 12,345
2 --> 24,690
4 --> 49,380
8 --> 98,760
16 --> 197,520
32 --> 395,040
64 --> 790,080
128 --> 1,580,160
256 --> 3,160,320
512 --> 6,320,640
1024 --> 12,641,280
2048 --> 25,282,560
4096 --> 50,565,120

---> Calculating the result
6,789 * 12,345 is: 50,565,120 + 25,282,560 + 6,320,640 + 1,580,160 + 49,380 + 12,345 = 83,810,205

Please enter operand1, or type "stop"
4
Please enter operand2
5
The smaller operand is: 4; and the larger operand is: 5
--> Creating the mapping table:
1 --> 5
2 --> 10
4 --> 20

---> Calculating the result
4 * 5 is: 20 = 20

Please enter operand1, or type "stop"
stop
Done multiplying

*** ELIMINATING ADJACENT DUPLICATES ***
--> testcase #1
input = [1, 5, 6, 8, 8, 8, 0, 1, 1, 0, 6, 5]
result = [1] CORRECT

---> testcase #2
input = [1, 9, 6, 8, 8, 8, 0, 1, 1, 0, 6, 5]
result = [1, 9, 5] CORRECT

---> testcase #3
input = [1, 1, 6, 8, 8, 8, 0, 1, 1, 0, 6, 5]
result = [5] CORRECT

---> testcase #4
input = [1, 1, 1, 5, 6, 8, 8, 8, 0, 1, 1, 0, 6, 5]
result = [] CORRECT
Done!

Process finished with exit code 0
```

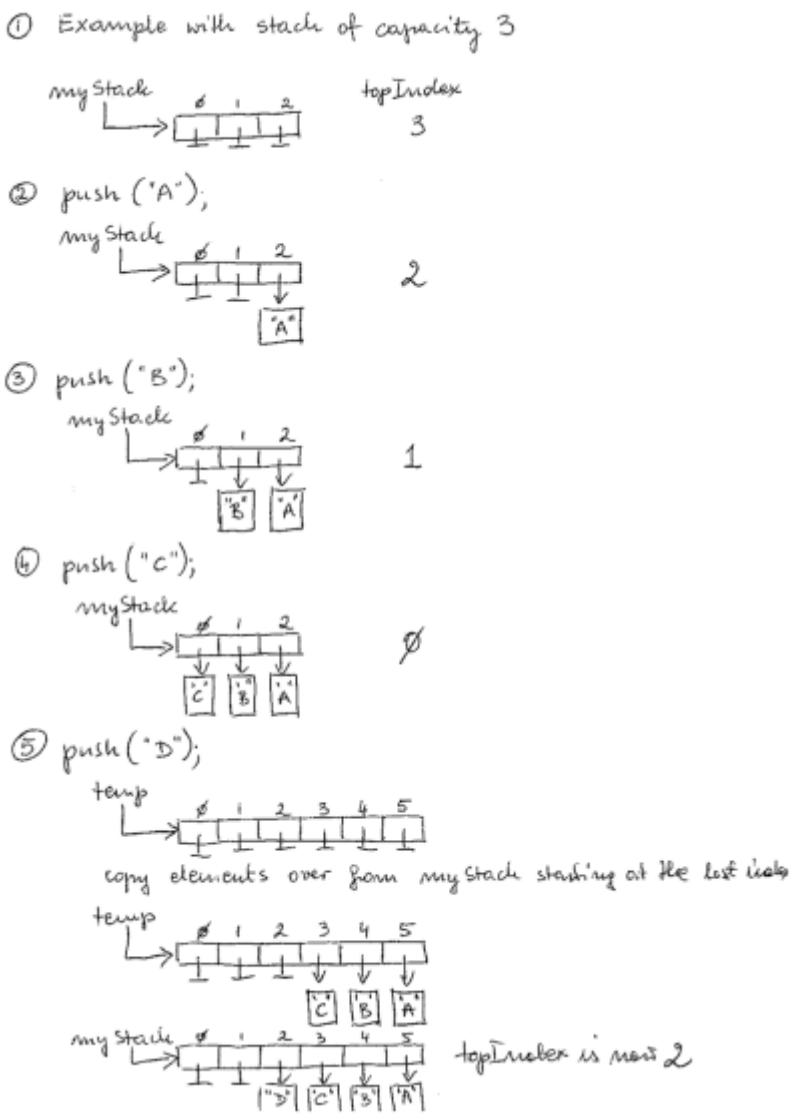
Application #2

`ArrayStack` class (provided as a skeleton) is to implement `TextbookStackInterface`

The instance variable `myStack` is defined as an array and it should contain stack's entries. The array should be expanded dynamically as necessary.

The instance variable `topIndex` is defined to keep the index of the stack's top entry.

When the stack is empty, the first object pushed on the stack will be at the last slot of the array. Maintain the stack's bottom entry in `this.myStack[this.myStack.length - 1]`. It means that the constructor must set `this.topIndex` to `this.mystack.length` also special care must be taken when array is resized. See the example below:



The provided `ArrayStack` class includes `main` with test cases to test your methods.

SAMPLE RUN:

```
*** Creating a stack with default constructor ***
---> The stack capacity is set by the constructor to: 50
---> The topIndex is: 50

---> isEmpty() returns true
CORRECT - the top index is set to 50

---> Adding to stack to get: Joe Jane Jill Jess Jim
---> Done adding 5 elements; the topIndex is: 45
---> isEmpty() returns false

--> Testing peek and pop:
Joe is at the top of the stack.
Joe is removed from the stack.
Jane is at the top of the stack.
Jane is removed from the stack.
Jill is at the top of the stack.
Jill is removed from the stack.
Jess is at the top of the stack.
Jess is removed from the stack.
Jim is at the top of the stack.
Jim is removed from the stack.
--> The stack should be empty:
isEmpty() returns true

--> Adding to stack to get: Jim Jess Joe
---> Done adding 3 elements; the topIndex is: 47

--> Testing clear:
--> The stack should be empty:
isEmpty() returns true
defaultStack.peek() returns:
 CORRECT - exception has been thrown: peek operation failed
defaultStack.pop() returns:
 CORRECT - exception has been thrown: pop operation failed


*** Creating a stack with the secondary constructor ***
---> The stack capacity is set by the constructor to: 3
---> The topIndex is: 3

--> isEmpty() returns true

--> Adding to stack to get: Joe Jane Jill Jess Jim
-----> The stack capacity has been doubled and it is now: 6; with topIndex = 3
---> Done adding 5 elements; the topIndex is: 1
--> isEmpty() returns false

-->Testing peek and pop:
Joe is at the top of the stack.
Joe is removed from the stack.
Jane is at the top of the stack.
Jane is removed from the stack.
Jill is at the top of the stack.
Jill is removed from the stack.
Jess is at the top of the stack.
Jess is removed from the stack.
Jim is at the top of the stack.
Jim is removed from the stack.
--> The stack should be empty:
isEmpty() returns true
CORRECT - the top index is set to 6

--> Adding to stack to get: Jim Jess Joe
---> Done adding 3 elements; the topIndex is: 3

--> Testing clear:
--> The stack should be empty:
```

```
isEmpty() returns true
smallStack.peek() returns:
 CORRECT - exception has been thrown: peek operation failed
smallStack.pop() returns:
 CORRECT - exception has been thrown: pop operation failed
*** Done ***
```

Application #3

The ADT stack lets you peek at its top entry without removing it. For some applications of stacks, you also need to peek at the entry beneath the top entry without removing it. We will call such an operation **peek2**:

- If the stack has more than one entry, `peek2` method returns the second entry from the top without altering the stack.
- If the stack has fewer than two entries, `peek2` throws `InsufficientNumberOfElementsOnStackException.`

Write a linked implementation of a stack that includes a method `peek2`
Skeleton of `LinkedStack` class is provided. The class includes `main` with test cases to test your methods.

SAMPLE RUN

```
*** Create a stack ***
--> Add to stack to get: Joe Jane Jill Jess Jim

Done adding 5 elements.

--> Testing peek, peek2, and pop:
Joe is at the top of the stack.
Jane is just beneath the top of the stack.
Joe is removed from the stack.

Jane is at the top of the stack.
Jill is just beneath the top of the stack.
Jane is removed from the stack.

Jill is at the top of the stack.
Jess is just beneath the top of the stack.
Jill is removed from the stack.

Jess is at the top of the stack.
Jim is just beneath the top of the stack.
Jess is removed from the stack.

Jim is at the top of the stack.
 CORRECT - exception has been thrown: cannot complete peek2() - only one element on the stack
Jim is removed from the stack.

--> The stack should be empty:
isEmpty() returns true
 CORRECT - exception has been thrown: cannot complete peek() - stack is empty
 CORRECT - exception has been thrown: cannot complete pop() - stack is empty
 CORRECT - exception has been thrown: cannot complete peek2() - stack is empty
*** Done ***
```

Application #4

Write a Java program that uses a stack to test whether an input string is a palindrome. A palindrome is a string of characters (a word, phrase, or sentence) that is the same regardless of whether you read it forward or backward if you ignore spaces, punctuation, and case. Design and implement an algorithm that utilizes one stack and no equals to check this property. HINT: you should push half of the characters on the stack.

Skeleton of `PalindromeChecker` class is provided.

SAMPLE RUN

```
*** This program determines whether a string is a palindrome.
A palindrome is spelled the same from left to right as it is from right to left,
if we ignore punctuation, spaces, and case. ***

Enter a string that you want to check (or enter "stop" to stop):
Race car
---> Checking: "racecar"
"Race car" is a palindrome!

Enter a string that you want to check (or enter "stop" to stop):
A man, a plan, a canal: Panama!
---> Checking: "amanaplanacanalpanama"
"A man, a plan, a canal: Panama!" is a palindrome!

Enter a string that you want to check (or enter "stop" to stop):
abracadabra
---> Checking: "abracadabra"
"abracadabra" is not a palindrome!

Enter a string that you want to check (or enter "stop" to stop):
stop
Done!
```

Application #5

Write an application that converts infix expression to postfix and evaluates it.

Class Postfix includes `convertToPostfix` and `evaluatePostfix` methods that should implement algorithms given in Segments 5.16 and 5.18 respectively. Assume that the given algebraic expressions are syntactically correct. The `String` method `replaceAll`, `StringBuilder` class, and `Character.isLetter` method should be utilized.

Your Task:

1. Analyze provided skeleton, UML diagram and the sample run.
2. When working with the `Stack` use only the stack's "vanilla" methods: `push`, `pop`, and `peek`.

SAMPLE RUN

```
Converting infix expressions to postfix expressions:

Infix:   a+b
Postfix: ab+

Infix:   (a + b) * c
Postfix: ab+c*

Infix:   a * b / (c - d)
Postfix: ab*cd-/

Infix:   a / b + (c - d)
Postfix: ab/cd-+

Infix:   a / b + c - d
Postfix: ab/c+d-

Infix:   a^b^c
Postfix: abc^^

Infix:   (a^b)^c
Postfix: ab^c^

Infix:   a*(b/c+d)
Postfix: abc/d+*

Infix:   (a+b)/(c-d)
Postfix: ab+cd-/

Infix:   a/(b-c)*d
Postfix: abc-/d*

Infix:   a-(b/(c-d)*e+f)^g
Postfix: abcd-/e*f+g^-

Infix:   (a-b*c)/(d*e^f*g+h)
Postfix: abc*-def^*g*h+/


Evaluating postfix expressions with
a = 2, b = 3, c = 4, d = 5, e = 6
Assuming correct input!!!

ae+bd-/ : -4.0
abc*d*- : -58.0
abc-/d* : -10.0
ebca^*+d- : 49.0

Done.
```

Application #6

Part1

Implement the following algorithm (method `sort()`) that sorts entries on a stack: ***original*** stack contains entries that are not sorted. First create two empty stacks, ***destination*** and ***temp***. At any given time, ***destination*** stack must hold the entries in sorted order, with the smallest at the top of the stack. Move the top entry of ***original*** stack to ***destination*** stack. Pop and consider the top entry ***topO*** of ***original*** stack. Using a while loop pop entries of ***destination*** stack and push them onto ***temp*** stack until you reach the correct place to put ***topO***. Then push ***topO*** onto ***destination*** stack. Next using a while loop move all the entries from ***temp*** stack to ***destination*** stack. Repeat the process for as long as the original stack is not empty.

Part2

Now consider the following revision of the above algorithm (method `sortRevised()`) : after moving the top entry of ***original*** stack to ***destination*** stack, compare the new top entry ***topO*** of the ***original*** stack with the top entry of ***destination*** stack. Then either move entries from ***destination*** stack to ***temp*** stack or from ***temp*** stack to ***destination*** stack until you locate the correct position for ***topO***. Push ***topO*** onto ***destination*** stack. Continue until ***original*** stack is empty. Finally move any entries remaining in ***temp*** stack to ***destination*** stack.

Your finished program should produce the following output:

```
Setting the original stack to:
03 09 01 04 06 05 07 08 00 02

***Calling sort method***
push 02 from original to destination
Moving entries from destination to temp
push 00 to destination
Moving entries from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 02 from destination to temp
push 08 to destination
Moving entries from temp to destination
--> push 02 from temp to destination
--> push 00 from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 02 from destination to temp
push 07 to destination
Moving entries from temp to destination
--> push 02 from temp to destination
--> push 00 from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 02 from destination to temp
push 05 to destination
Moving entries from temp to destination
--> push 02 from temp to destination
--> push 00 from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 02 from destination to temp
--> push 05 from destination to temp
push 06 to destination
Moving entries from temp to destination
--> push 05 from temp to destination
--> push 02 from temp to destination
```

```
--> push 00 from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 02 from destination to temp
push 04 to destination
Moving entries from temp to destination
--> push 02 from temp to destination
--> push 00 from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
push 01 to destination
Moving entries from temp to destination
--> push 00 from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 01 from destination to temp
--> push 02 from destination to temp
--> push 04 from destination to temp
--> push 05 from destination to temp
--> push 06 from destination to temp
--> push 07 from destination to temp
--> push 08 from destination to temp
push 09 to destination
Moving entries from temp to destination
--> push 08 from temp to destination
--> push 07 from temp to destination
--> push 06 from temp to destination
--> push 05 from temp to destination
--> push 04 from temp to destination
--> push 02 from temp to destination
--> push 01 from temp to destination
--> push 00 from temp to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 01 from destination to temp
--> push 02 from destination to temp
push 03 to destination
Moving entries from temp to destination
--> push 02 from temp to destination
--> push 01 from temp to destination
--> push 00 from temp to destination

Stack should be sorted (with sort()) ....
00 01 02 03 04 05 06 07 08 09

==================================

Testing the revised method
Setting the original stack to:
03 09 01 04 06 05 07 08 00 02

***Calling sortRevised method***
--> push 02 from original to destination
Moving entries from destination to temp
Moving entries from temp to destination
push 00 from original to destination
Moving entries from destination to temp
--> push 00 from destination to temp
--> push 02 from destination to temp
Moving entries from temp to destination
push 08 from original to destination
Moving entries from destination to temp
Moving entries from temp to destination
push 07 from original to destination
Moving entries from destination to temp
Moving entries from temp to destination
push 05 from original to destination
Moving entries from destination to temp
--> push 05 from destination to temp
Moving entries from temp to destination
push 06 from original to destination
```

```
Moving entries from destination to temp
Moving entries from temp to destination
--> push 05 from temp to destination
push 04 from original to destination
Moving entries from destination to temp
Moving entries from temp to destination
--> push 02 from temp to destination
push 01 from original to destination
Moving entries from destination to temp
--> push 01 from destination to temp
--> push 02 from destination to temp
--> push 04 from destination to temp
--> push 05 from destination to temp
--> push 06 from destination to temp
--> push 07 from destination to temp
--> push 08 from destination to temp
Moving entries from temp to destination
push 09 from original to destination
Moving entries from destination to temp
Moving entries from temp to destination
--> push 08 from temp to destination
--> push 07 from temp to destination
--> push 06 from temp to destination
--> push 05 from temp to destination
--> push 04 from temp to destination
push 03 from original to destination
Moving any remaining entries from temp to destination
--> push 02 from temp to destination
--> push 01 from temp to destination
--> push 00 from temp to destination

Stack should be sorted (with sortRevised()) ....
00 01 02 03 04 05 06 07 08 09
```