STACKS & QUEUES

LINKED LIST IMPLEMENTATIONS OF STACKS & QUEUES

QUEUES

- **A queue is a data structure that stores a list of items**
- **Items are always added at the end of the list**
- **Items are always removed from the start of the list**

QUEUES

- **Queue Operations**
  - enqueue: add an item at the end of the queue
  - dequeue: remove the first item in the queue and return its value
  - Peek : return the value of the first item without removing it

QUEUES

- **Queue Operations**
  - enqueue: add an item at the end of the queue
  - dequeue: remove the first item in the queue and return its value
  - Peek : return the value of the first item without removing it
- **We implemented similar methods in the LinkedList class**
  - enqueue → add(int k)
  - dequeue → similar to removeFirst()
    - But we need to return the removed value
  - Peek → return head.getData()

```
public class Queue {
    //private variables:
    private IntNode head;
    private IntNode tail;

    //constructor
    public Queue(){
        //empty linked list
        head=null;
        tail=null;
    }
    //check if linked list is empty
    public boolean isEmpty(){
        if (head ==null)
            return true;
        else return false;
    }
```

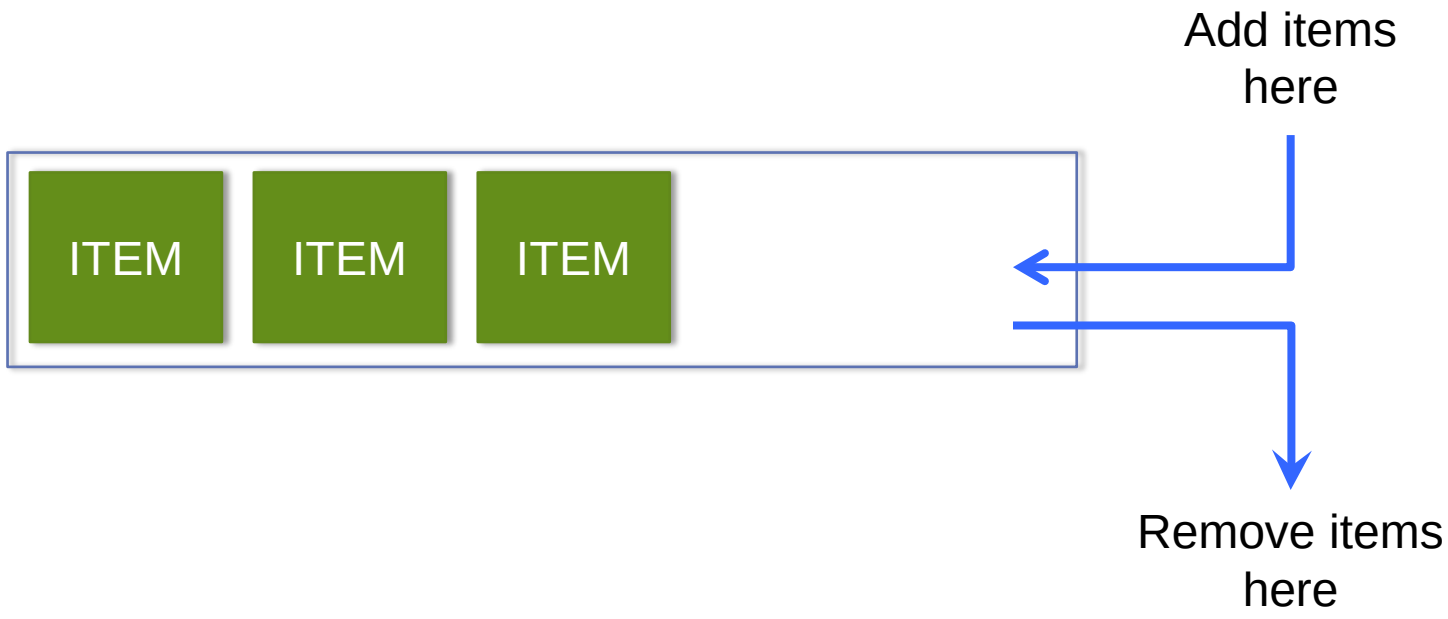
QUEUES

- **Make sure the Queue is not empty before using dequeue()**

```
If ( ! q.isEmpty() ) {
      q.dequeue();
}
```

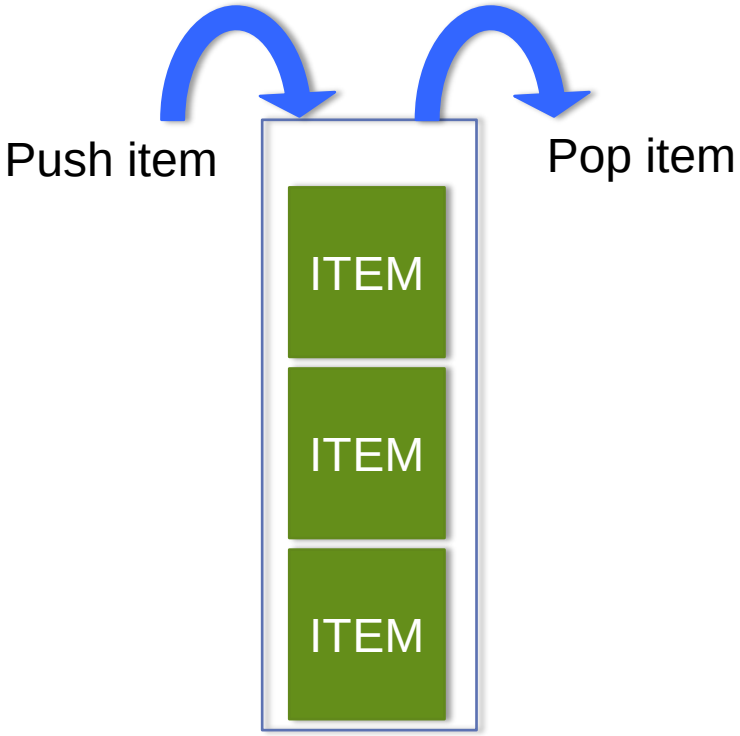
STACKS

- **A data structure that stores a list of items**
- **Items are always added at the end of the list**
- **Items are always removed from the end of the list**



STACKS

- **Usually visualized as a vertical stack**



STACKS

- **Stack Operations:**
  - Push : add an item at the top of the stack
  - Pop : remove an item from the top of the stack
  - Peek : check the value of the item at the top without removing it
- **We implemented similar methods in the LinkedList class**
  - Add → addFirst(int k)
  - Pop → similar to removeFirst()
    - But we need to return the removed value
  - Peek → return head.getData()

GENERIC TYPES

- **Using this IntNode class, we can only have a list of integers:**

```
public class IntNode {
    private int data;
    private IntNode next;
    private IntNode previous;

    //initialize node
    public IntNode(int value){
        data=value;
        next=null;
        previous=null;
    }
```

```
public class Queue {
    //private variables:
    private IntNode head;
    private IntNode tail;
```

GENERIC TYPES

- **Using a genetic class, we can have an item of any type**

```
public class ListNode<T> {
    private T data;
    private ListNode<T> next;
    private ListNode<T> previous;

    //initialize node
    public ListNode(T value){
        data=value;
        next=null;
        previous=null;
    }
```

GENERIC TYPES

- **All methods that reference the data items are replaced by T, which is the type parameter**

```
//return the data value
public T getData(){
    return data;
}
//return the next node
public ListNode<T> getNext(){
    return next;
}
```

GENERIC TYPES

- **Queue should also be generic**

```
public class Queue<T> {
    //private variables:
    private ListNode<T> head;
    private ListNode<T> tail;

    //add a node to the list
    public void enqueue(T value){

    }
    //remove first and return its value
    public T dequeue(){
```

GENERIC TYPES

- **The type is specified when an object is created:**

```
Queue<Integer> q=new Queue<Integer>;
q.add(3);

Queue<String> q2=new Queue<String>;
q2.add(“hello”);
```

APPLICATION: BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets) and check if they are balanced] or not

- **This can be implemented using a Stack. How?**

APPLICATION: BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets) and check if they are balanced] or not

- **This can be implemented using a Stack. How?**
  - Read characters one at a time
  - Push opening brackets into a stack
  - When you see a closing brackets, pop a character from the stack
  - If the character matches, continue, otherwise return an error.
  - At the end of the string, if the stack is not empty, return an error

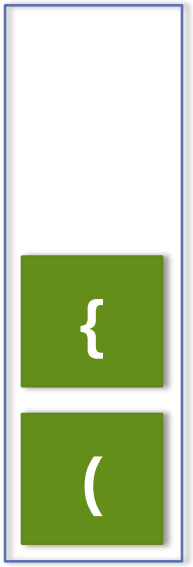
APPLICATION: BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses **(**and {braces} and brackets) and check if they are balanced] or not
  - PUSH the opening parenthesis into the stack

Stack

APPLICATION: BALANCED BRACKETS

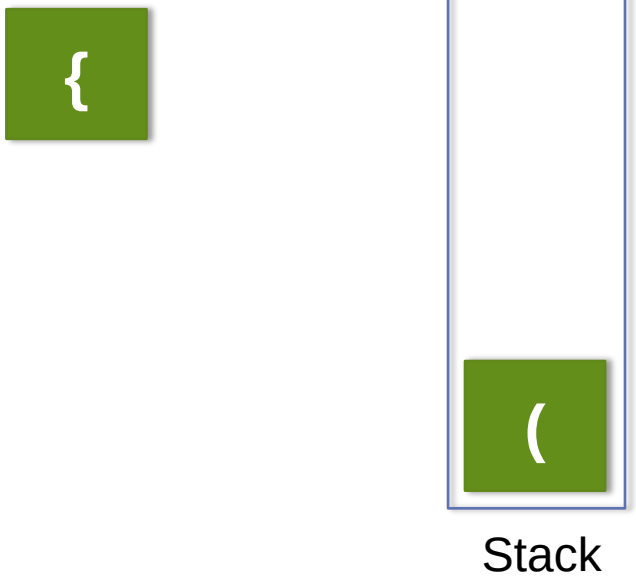
- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets) and check if they are balanced] or not
  - PUSH the opening braces into the stack



Stack

APPLICATION:
BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets) and check if they are balanced] or not
  - POP an item from the stack



APPLICATION: BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets) and check if they are balanced] or not
  - It matches, continue

Stack

APPLICATION:
BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets**)** and check if they are balanced] or not
  - POP an item from the stack **(**

APPLICATION: BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets) and check if they are balanced] or not
  - It matches, continue

Stack

APPLICATION: BALANCED BRACKETS

- **Check if the brackets, parentheses, and braces in a string are balanced:**
  - Example:
    Find all parentheses (and {braces} and brackets) and check if they are balanced] or not
  - Stack is empty, no balancing bracket.

Stack

APPLICATION: POSTFIX EXPRESSIONS

- **Numbers followed by operators. No need for parentheses.**

INFIX EXPRESSION	POSTFIX EXPRESSION
3 + 4	3 4 +
(3 + 4) * 5	3 4 + 5 *
(3 + 4) * (5 – 2)	3 4 + 5 2 - *

APPLICATION: POSTFIX EXPRESSIONS

- **Postfix expressions can be evaluated using a stack.**
  - Push numbers in the stack
  - When an operator is encountered, pop two numbers from the stack
  - Apply the operation on the two numbers
  - Push the result back into the stack

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 4 + 5 2 - *

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - **3** 4 + 5 2 - *
  - PUSH 3 into the stack

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4** + 5 2 - *
  - PUSH 4 into the stack

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4** + 5 2 - *
  - POP two items and calculate

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4** + 5 2 - *
  - PUSH result back into the stack

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4** **+** **5** 2 - *
  - PUSH 5 into the stack

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4 + 5 2** - *
  - PUSH 2 into the stack

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4 + 5 2 -** *
  - POP two items and calculate

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4 + 5 2 -** *
  - PUSH result back into the stack

APPLICATION: POSTFIX EXPRESSIONS

- **Example: Evaluate the following postfix expression:**
  - 3 **4 + 5 2 - ***
  - POP two items and calculate