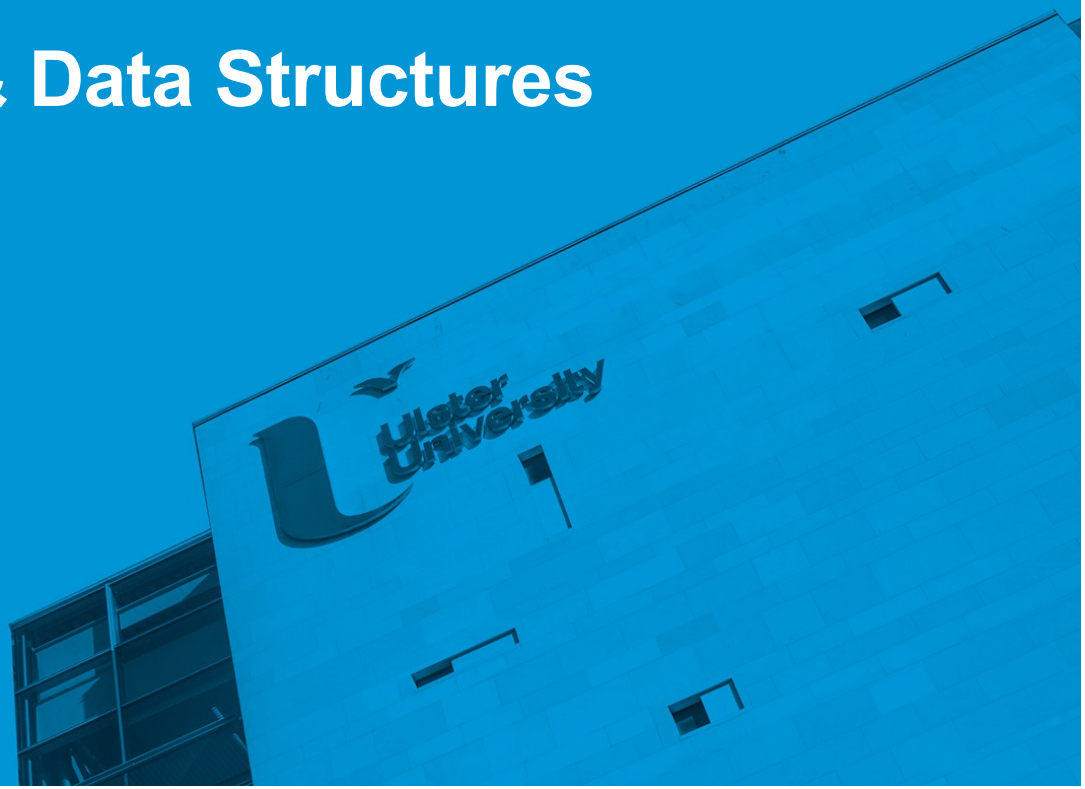
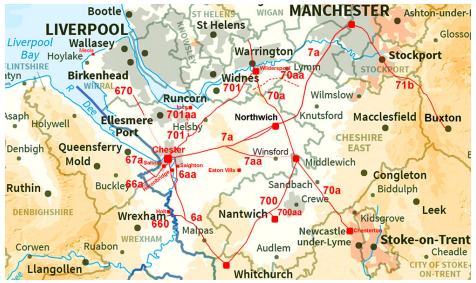
Ulster University

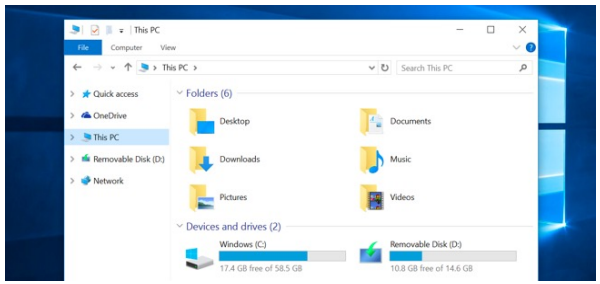
COM498 Algorithms & Data Structures

3.3 Stacks



Recall Examples of Data Organisation

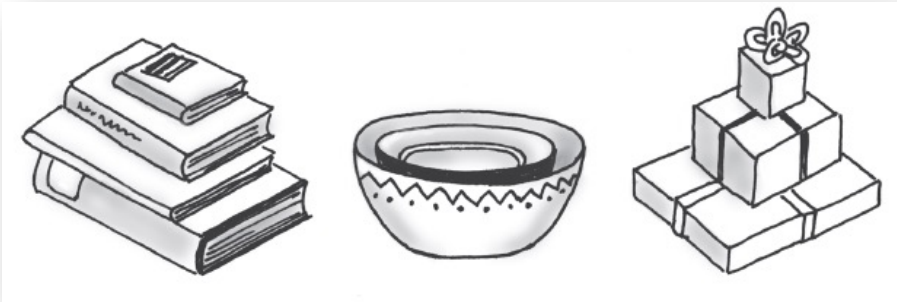




- Stack – Last in, first out – a very common data organisation technique in everyday life

Stack

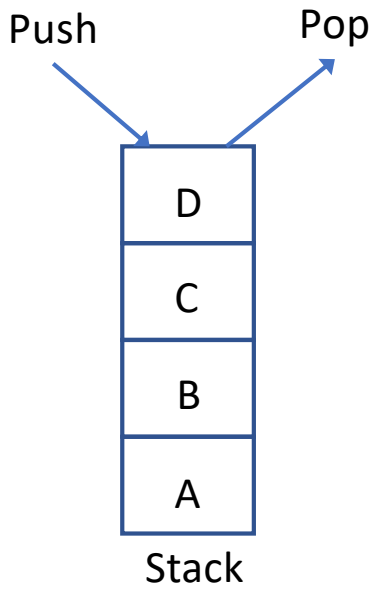
- In everyday life a stack is a familiar thing (stack of books, stack of dishes, stack of presents . . .)
- When you remove an item, you take the one on the top of the stack
- Topmost item is the last one that was added to the stack



Stack Operations

- The behaviour of a stack is also known as: **LIFO** (Last In, First Out)
- LIFO is exactly the behavior required by many important algorithms
- Such algorithms make use of the Stack ADT

- In a stack all additions are to one end of the stack called the **top** (the entry at the top is the newest item in a stack)
- The operation that adds an entry to the stack is traditionally called **push**
- The operation that removes an entry from the stack is traditionally called **pop**



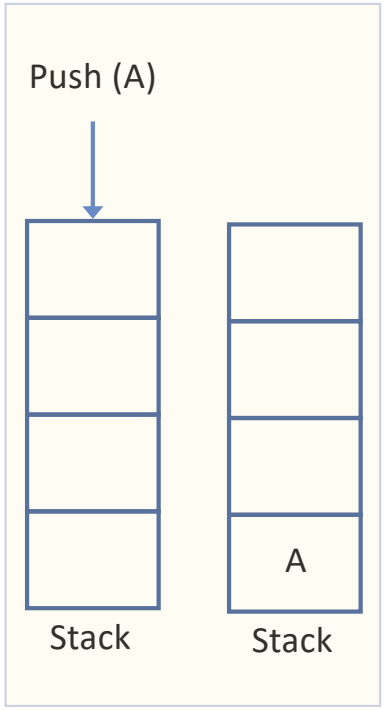
Stack Operations

Ulster University

- The stack restricts access to its entries (can only look at or remove top entry)
- In addition to push and pop, the operation to retrieve the top entry without removing it is called **peek**

- Typically you cannot search a stack for a specific entry
- The only way to look at an entry not at the top of the stack is to repeatedly remove items from the stack until the desired item reaches the top

Stack Operations



Design Decision

Ulster University

- When the stack is empty:
  - What to do with **pop** and **peek**?

- Possible actions:
  - Assume that the ADT is not empty (enforce a precondition)
  - Return `null` (okay as long as the ADT doesn't permit null entries)
  - Throw an exception (which type of exception? Checked or Runtime)

As calling pop or peek when a stack is empty is considered as a mistake by the client, a runtime exception should be thrown and can be handled by the application

Java Interface for the Stack ADT

```
public interface StackInterface<T> {
    public void push (T newEntry);
    public T pop();
    public T peek();
    public boolean isEmpty();
    public void clear();
}
```

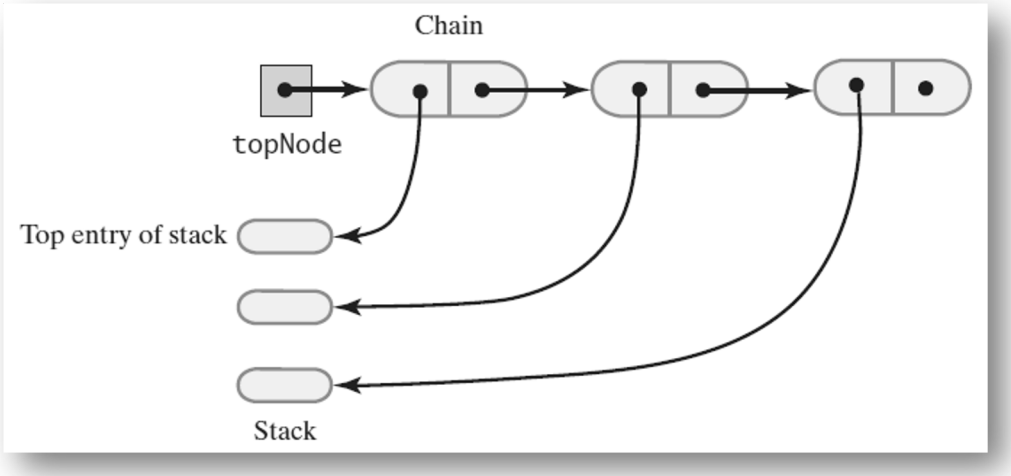
- `push` — Add a new entry to the top of the stack
- `pop` — Remove entry from the top of the stack
- `peek` — Return entry from the top of the stack
- `isEmpty` — Check for no entries in stack
- `clear` — Remove all entries from the stack

Scenario

- Create a new project called LinkedList, in which we will implement various experiments with linked structures
- Copy the file *MyNode.java* ifrom your Bag project into the **src** folder of the LinkedList project
- Create the file *StackInterface.java* and implement the interface class `StackInterface`

Linked Chain Implementation

- If we use a chain of linked nodes to implement a stack, where in the chain should we place the stack's top entry?



Since ALL stack activity is with the top element, the first node in the chain should reference the top entry of the stack

- Using a head reference, the first node can be added, removed and accessed faster than other nodes

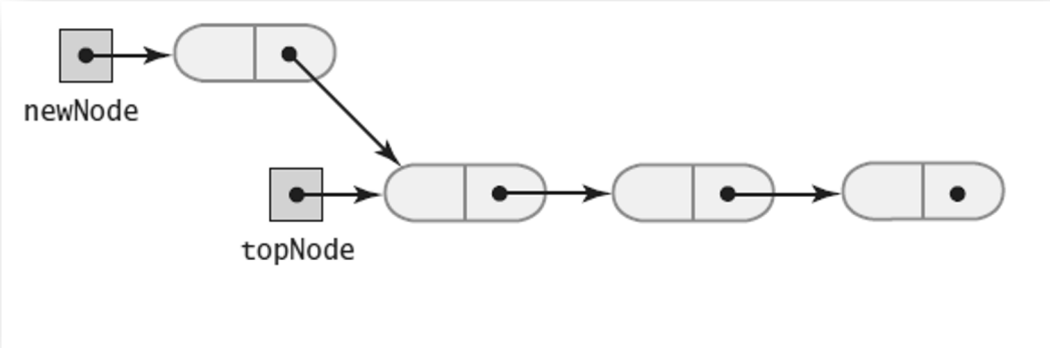
Linked Chain Implementation

Ulster University

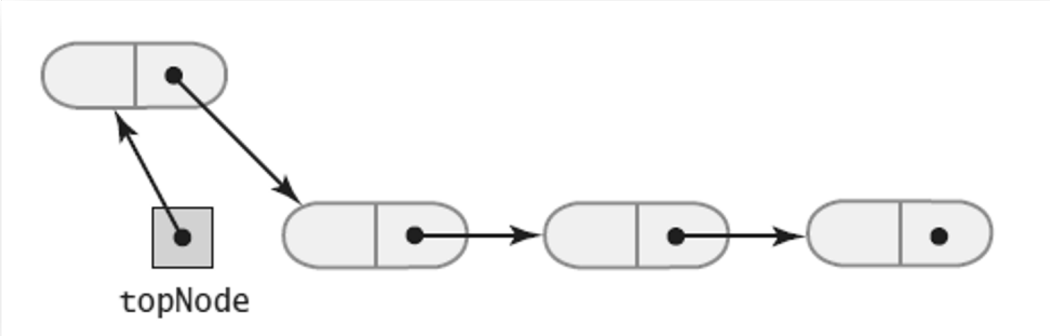
- Maintain pointer `topNode` to the first node in a singly-linked chain

```
public class Stack<T> implements StackInterface<T> {

   private MyNode<T> topNode;

   public Stack() { topNode = null; }

   // methods push(), pop(), peek(), isEmpty(), clear()
   // as defined in the interface
}
```

Linked Implementation of `push()`

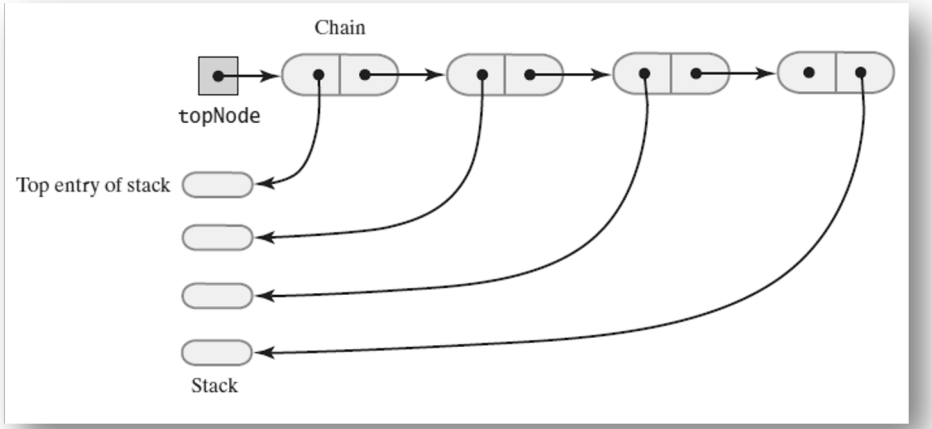


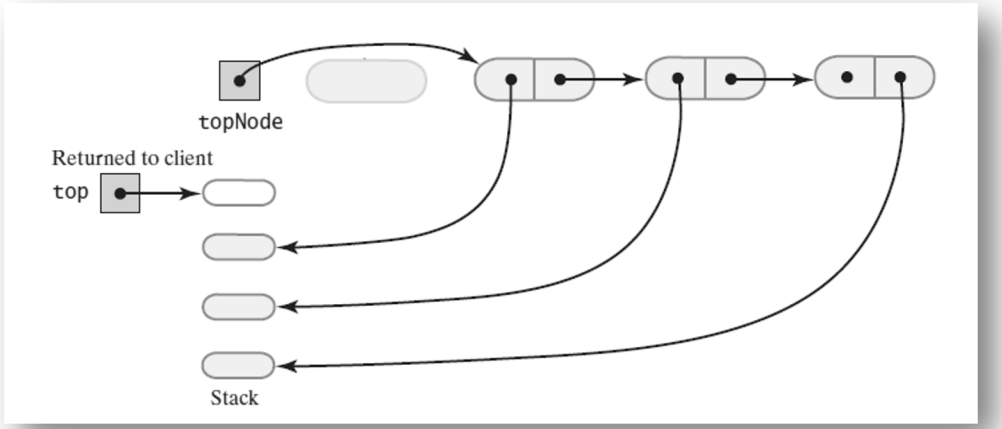
- An entry is pushed onto the stack by first allocating a new node (`newNode`) that references the stack's existing chain (`next` field points to the head reference `topNode`)



- The head reference of the chain (`topNode`) is then set to reference the new node (`newNode`)

Linked Implementation of `pop()`





- The top entry in the stack is obtained by accessing the data portion of the first node in the chain (through `topNode`)
- The top entry is removed by setting `topNode` to the reference in the first node. `topNode` will now reference what was originally the second node in the chain (or `null`)

Scenario

Ulster University

- Create the file *Stack.java* and provide the code for the class `Stack` as an implementation of `StackInterface` that uses a singly-linked list as the data organisation technique
  - Define the `topNode` instance variable that points to the first element in the chain
  - Define the constructor that creates a new stack object
  - Provide the implementation of all methods specified in the `StackInterface` class

- Check your implementation by providing a `main()` method that:
  - Creates a stack of `Integer` objects and pushes 3 values onto the stack
  - Attempts to peek and then pop 4 values from the stack
  - Pushes another 3 values to the stack and checks for an empty stack
  - Clears the stack and repeats the check for an empty stack

Stack Applications Example

- In an algebraic expression (with no parentheses) operators occur in a certain order (exponential, multiply and divide, add and subtract)

  20 – 2 * 2 ^ 3 evaluates as 20 – 2 * 8 then as 20 – 16 = 4

- What if two or more adjacent operators have same precedence? (different for exponentiation than for other operators)

What does 2 ^ 2 ^ 3 evaluate as?	2 ^ 2 ^ 3 evaluates as 2 ^ (2 ^ 3), so 2 ^ 8 = 256
What does 8 - 4 + 2 evaluate as?	8 – 4 + 2 evaluates as (8 – 4) + 2, so 4 + 2 = 6

Stack Applications Example

Ulster University

- **Arithmetic notations**
  - Infix: each binary operator appears between its operands: ***a + b***
  - Prefix: each binary operator appears before its operands: ***+ a b***
  - Postfix: each binary operator appears after its operands: ***a b +***

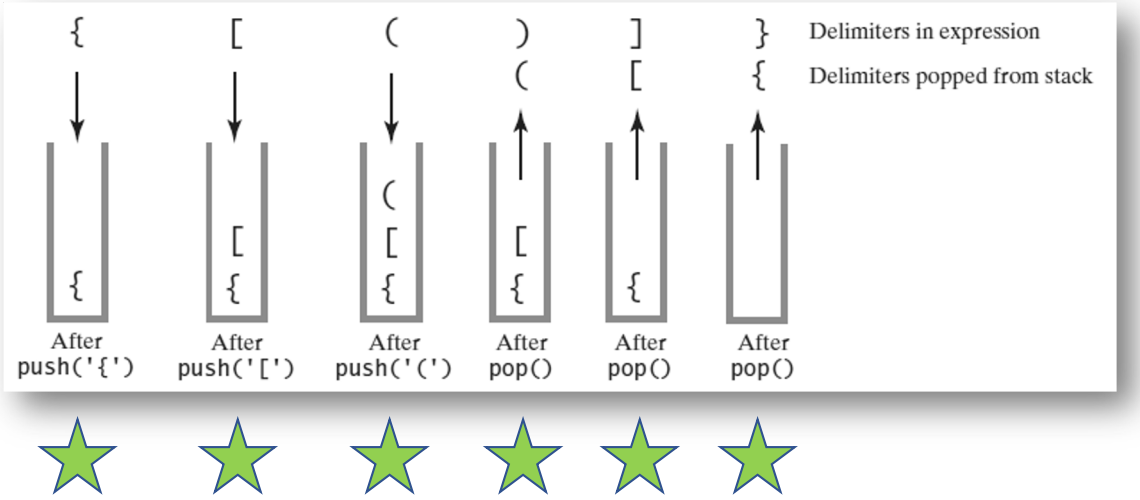
prefix and postfix are easier to process – don't need precedence rules or parentheses.
For example **a + (b – c)** would be written in postfix as **a b c - +**

- When we use parentheses they must be paired correctly!
  - An open parenthesis must correspond to a closed parenthesis
  - Pairs or parentheses must not intersect
- Balanced expressions: delimiters paired correctly (are balanced), i.e. **{ [( ) ( )] ( ) }**

Check For Balanced Expressions

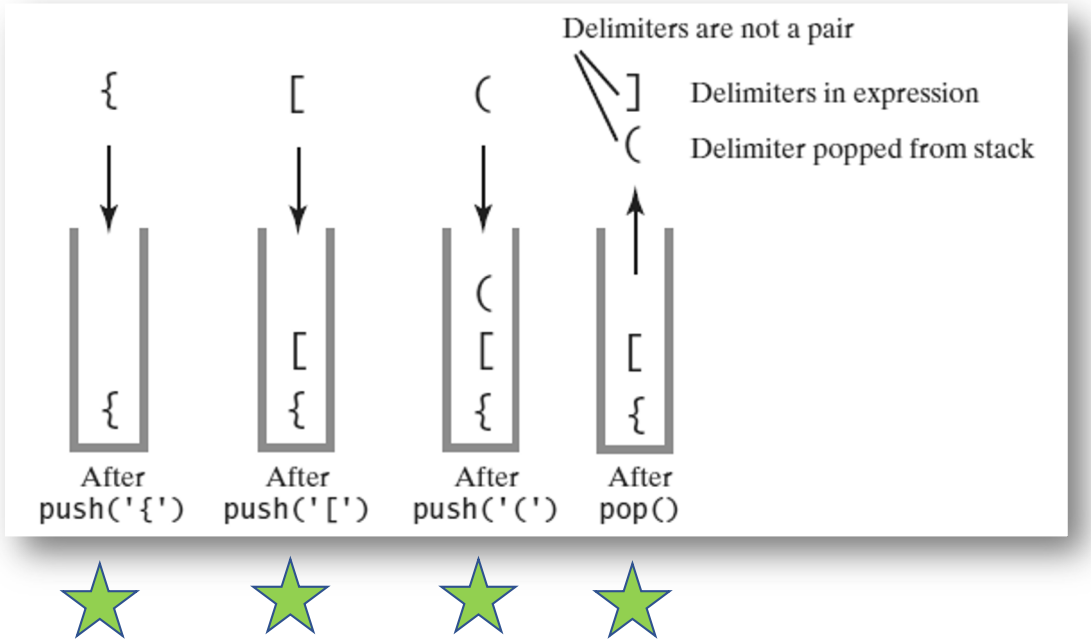
- We can use a stack to check if an expression has paired delimiters
- Check if the expression **a { b [c (d + e) / 2 – f] + 1 }** is a balanced expression, we scan the expression from left-to-right and ignore any characters that are not delimiters

- When we encounter an open delimiter push it onto the stack
- When we find a close delimiter see if it corresponds to the open delimiter at the top of the stack
- If it does, then pop the open delimiter from the top of the stack and continue



Check For Balanced Expressions

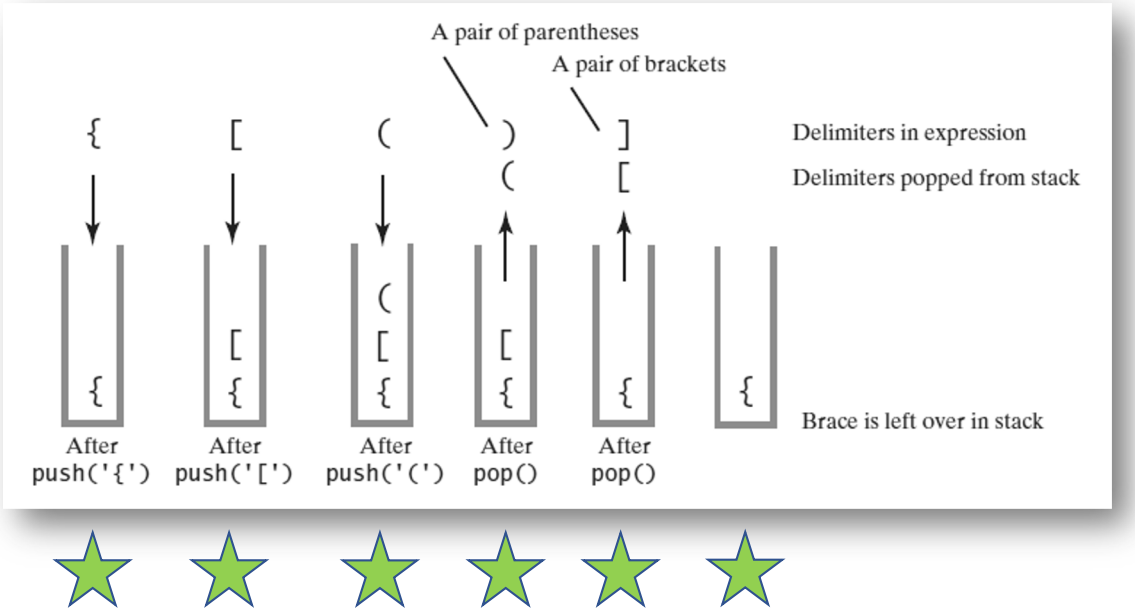
- Check if the expression **a { b [c (d + e] / 2 – f) + 1 }** is a balanced expression



- Push the first three open delimiters onto the stack
- Next delimiter scanned in the expression is a close bracket **]**
- Open delimiter **(** at the top of the stack does not match, so expression is **not balanced**

Check For Balanced Expressions

- Check if the expression **a { b [c (d + e) / 2 – f] + 1** is a balanced expression



- Push the first three open delimiters onto the stack
- Next delimiter scanned in the expression is a close parenthesis **)** which matches the top of the stack, so pop the top of the stack
- Next delimiter is a close bracket **]** which matches top of the stack, so pop the top of the stack
- End of expression reached and stack still contains open brace **{** so expression is **not balanced**

Check For Balanced Expressions

```
Algorithm checkBalance(expression)
// Return true if an expression is balanced, false otherwise

set isBalanced to true
while isBalanced and not end of expression
   if next character in expression is (, [ or {
      push character onto stack
   else if next character in expression is ), ] or }
      if stack is empty set isBalanced to false
      else set openDelimiter to value popped from stack
           if openDelimiter and next character in expression are not a matching pair
              set isBalanced to false
end while
if stack is not empty set isBalanced to false
return isBalanced
```

Scenario

- Create the new file *BalancedExpression.java* in your LinkedList project and implement the application class `BalancedExpression`
  - The application class should contain a `main()` method that prompts the user for an expression using **( )**, **[ ]** and **{ }** as delimiters for elements of the expression. The expression should contain no spaces and all other characters are assumed to be single character symbols
  - When the expression is entered by the user, the `main()` method should call a `balancedExpression()` method that returns true if the expression is balanced or false otherwise
  - The `main()` method should then report whether or not the expression is balanced

Java Class Library: The Class `Stack`

- Found in package `java.util`

java.util

Class Stack<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.Vector<E>
        java.util.Stack<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Constructor Summary

Constructors

Constructor and Description
`Stack()` Creates an empty Stack.

Method Summary

Methods

Modifier and Type	Method and Description
`boolean`	`empty()` Tests if this stack is empty.
`E`	`peek()` Looks at the object at the top of this stack without removing it from the stack.
`E`	`pop()` Removes the object at the top of this stack and returns that object as the value of this function.
`E`	`push(E item)` Pushes an item onto the top of this stack.
`int`	`search(Object o)` Returns the 1-based position where an object is on this stack.

Challenge

- In your **LinkedList** project, create the file *Infix2Postfix.java* and implement the class `Infix2Postfix` to to house an application that accepts an infix expression from the keyboard as input, converts it to postfix notation, outputs the postfix string and then calculates and outputs the result.
- Use the algorithms for infix to postfix conversion and postfix evaluation found in the Infix2Postfix Challenge document found on Blackboard.