|  |  |
| --- | --- |
| Bracket Balancing: |  |
| Stack | 1. linear data structure which follows a particular order. 2. The order may be LIFO(Last In First Out) or FILO(First In Last Out). 3. **push(), pop(), isEmpty() and peek()** all take O(1) time. 4. We do not run any loop in any of these operations. |
| **Applications of stack:** | 1. Balancing of symbols. 2. Infix to Postfix /Prefix conversion. 3. Redo-undo features at many places like editors, photoshop. 4. Forward and backward feature in web browsers. 5. Used in many algorithms like Tower of Hanoi, tree traversals, stock span problem, histogram problem. 6. Other applications can be Backtracking, Knight tour problem, rat in a maze, N queen problem and sudoku solver. 7. In Graph Algorithms like Topological Sorting and Strongly Connected Components. |
| **Implementation:** | Using array  Using linked list |
|  | The order is First In First Out (FIFO). |
|  |  |
|  |  |
|  |  |
| **Implementing Stack using Arrays** | **class** StackDS {  **static** **final** **int** ***MAX*** = 1000;  **int** top;  **int** a[] = **new** **int**[***MAX***]; // Maximum size of Stack    StackDS()  {  top = -1;  }  **boolean** isEmpty()  {  **return** (top < 0);  }    **boolean** push(**int** x)  {  **if** (top >= (***MAX*** - 1)) {  System.***out***.println("Stack Overflow");  **return** **false**;  }  **else** {  a[++top] = x;  System.***out***.println(x + " pushed into stack");  **return** **true**;  }  }  **int** pop()  {  **if** (top < 0) {  System.***out***.println("Stack Underflow");  **return** 0;  }  **else** {  **int** x = a[top--];  **return** x;  }  }  **int** peek()  {  **if** (top < 0) {  System.***out***.println("Stack Underflow");  **return** 0;  }  **else** {  **int** x = a[top];  **return** x;  }  }  } |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |