* [DP Stories](http://www.programcreek.com/java-design-patterns-in-stories/) (Stories to remember design patterns )

[**Top 10 Algorithms for Coding Interview**](http://www.programcreek.com/2012/11/top-10-algorithms-for-coding-interview/)

Lastest Upate: 3/20/2015 ([PDF Version](http://www.programcreek.com/wp-content/uploads/2012/11/coding-interview1.pdf))

The following are the common topics for coding interviews. As understanding those concepts requires much more effort, this tutorial only serves as an introduction. The topics that are covered include: *1) String/Array/Matrix, 2) Linked List, 3) Tree, 4) Heap, 5) Graph, 6) Sorting, 7) Recursion vs. Iteration, 8) Dynamic Programming, 9) Bit Manipulation, 10) Probability, 11) Combinations and Permutations, and other interesting problems.* I highly recommend you to read ["Simple Java"](http://www.programcreek.com/simple-java/) first, if you need a brief review of Java basics. If you want to see examples/projects using a library, you can use [JavaSED.com](http://www.javased.com/).

Common Approaches to solve problems: *sorting*, *binary search*, *hash table*, *heap*, *tree*, *depth-first search*, *dynamic programming*.

**1. String/Array/Matrix**

String in Java is a class that contains a char array and other fields and methods. Without auto-completion from any IDE, the following methods should be remembered.

|  |
| --- |
| toCharArray() *//get char array of a String*  charAt(**int** x) *//get a char at the specific index*  length() *//string length*  length *//array size*  substring(**int** beginIndex)  substring(**int** beginIndex, **int** endIndex)  Integer.valueOf()*//string to integer*  String.valueOf()/integer to string  Arrays.sort() *//sort an array*  Arrays.toString(**char**[] a) *//convert to string*  Arrays.copyOf(T[] original, **int** newLength)  System.arraycopy(Object src, **int** srcPos, Object dest, **int** destPos, **int** length) |

Strings/arrays are easy to understand, but the interview problems often require advanced algorithm to solve, such as dynamic programming, recursion, etc.

Classic problems:  
[0) Rotate Array](http://www.programcreek.com/2015/03/rotate-array-in-java/)  
[1) Evaluate Reverse Polish Notation (Stack)](http://www.programcreek.com/2012/12/leetcode-evaluate-reverse-polish-notation/)  
[2) Longest Palindromic Substring (DP)](http://www.programcreek.com/2013/12/leetcode-solution-of-longest-palindromic-substring-java/)  
[3) Word Break (DP)](http://www.programcreek.com/2012/12/leetcode-solution-word-break/)  
[3) Word Break II (DP, DFS)](http://www.programcreek.com/2014/03/leetcode-word-break-ii-java/)  
[4) Word Ladder (Queue, BFS)](http://www.programcreek.com/2012/12/leetcode-word-ladder/)  
[5) Median of Two Sorted Arrays](http://www.programcreek.com/2012/12/leetcode-median-of-two-sorted-arrays-java/)  
[6) Regular Expression Matching](http://www.programcreek.com/2012/12/leetcode-regular-expression-matching-in-java/)  
[7) Merge Intervals](http://www.programcreek.com/2012/12/leetcode-merge-intervals/)  
[8) Insert Interval](http://www.programcreek.com/2012/12/leetcode-insert-interval/)  
[9) Two Sum](http://www.programcreek.com/2012/12/leetcode-solution-of-two-sum-in-java/)  
[9) Two Sum II – Input array is sorted](http://www.programcreek.com/2014/03/two-sum-ii-input-array-is-sorted-java/)  
[9) Two Sum III - Data structure design](http://www.programcreek.com/2014/03/two-sum-iii-data-structure-design-java/)  
[9) 3Sum](http://www.programcreek.com/2012/12/leetcode-3sum/)  
[9) 4Sum](http://www.programcreek.com/2013/02/leetcode-4sum-java/)  
[10) 3Sum Closest](http://www.programcreek.com/2013/02/leetcode-3sum-closest-java/)  
[11) String to Integer](http://www.programcreek.com/2012/12/leetcode-string-to-integer-atoi/)  
[12) Merge Sorted Array](http://www.programcreek.com/2012/12/leetcode-merge-sorted-array-java/)  
[13) Valid Parentheses](http://www.programcreek.com/2012/12/leetcode-valid-parentheses-java/)  
[14) Implement strStr()](http://www.programcreek.com/2012/12/leetcode-implement-strstr-java/)  
[15) Set Matrix Zeroes](http://www.programcreek.com/2012/12/leetcode-set-matrix-zeroes-java/)  
[16) Search Insert Position](http://www.programcreek.com/2013/01/leetcode-search-insert-position/)  
[17) Longest Consecutive Sequence](http://www.programcreek.com/2013/01/leetcode-longest-consecutive-sequence-java/)  
[18) Valid Palindrome](http://www.programcreek.com/2013/01/leetcode-valid-palindrome-java/)  
[19) Spiral Matrix](http://www.programcreek.com/2013/01/leetcode-spiral-matrix-java/)  
[20) Search a 2D Matrix](http://www.programcreek.com/2013/01/leetcode-search-a-2d-matrix-java/)  
[21) Rotate Image](http://www.programcreek.com/2013/01/leetcode-rotate-image-java/) [Palantir]  
[22) Triangle](http://www.programcreek.com/2013/01/leetcode-triangle-java/)  
[23) Distinct Subsequences Total](http://www.programcreek.com/2013/01/leetcode-distinct-subsequences-total-java/)  
[24) Maximum Subarray](http://www.programcreek.com/2013/02/leetcode-maximum-subarray-java/) [Palantir, LinkedIn]  
[24) Maximum Product Subarray](http://www.programcreek.com/2014/03/leetcode-maximum-product-subarray-java/) [LinkedIn]  
[25) Remove Duplicates from Sorted Array](http://www.programcreek.com/2013/01/leetcode-remove-duplicates-from-sorted-array-java/)  
[26) Remove Duplicates from Sorted Array II](http://www.programcreek.com/2013/01/leetcode-remove-duplicates-from-sorted-array-ii-java/)  
[27) Longest Substring Without Repeating Characters](http://www.programcreek.com/2013/02/leetcode-longest-substring-without-repeating-characters-java/)  
[28) Longest Substring that contains 2 unique characters](http://www.programcreek.com/2013/02/longest-substring-which-contains-2-unique-characters/) [Google]  
[29) Palindrome Partitioning](http://www.programcreek.com/2013/03/leetcode-palindrome-partitioning-java/)  
[30) Reverse Words in a String](http://www.programcreek.com/2014/02/leetcode-reverse-words-in-a-string-java/)   
[31) Find Minimum in Rotated Sorted Array](http://www.programcreek.com/2014/02/leetcode-find-minimum-in-rotated-sorted-array/)   
[31) Find Minimum in Rotated Sorted Array II](http://www.programcreek.com/2014/03/leetcode-find-minimum-in-rotated-sorted-array-ii-java/)  
[32) Find Peak Element](http://www.programcreek.com/2014/02/leetcode-find-peak-element/)  
[33) Min Stack](http://www.programcreek.com/2014/02/leetcode-min-stack-java/)  
[34) Majority Element](http://www.programcreek.com/2014/02/leetcode-majority-element-java/)  
[35) Combination Sum (DFS)](http://www.programcreek.com/2014/02/leetcode-combination-sum-java/)  
[36) Best Time to Buy and Sell Stock](http://www.programcreek.com/2014/02/leetcode-best-time-to-buy-and-sell-stock-java/)   
[36) Best Time to Buy and Sell Stock II](http://www.programcreek.com/2014/02/leetcode-best-time-to-buy-and-sell-stock-ii-java/)  
[36) Best Time to Buy and Sell Stock III (DP)](http://www.programcreek.com/2014/02/leetcode-best-time-to-buy-and-sell-stock-iii-java/)  
[36) Best Time to Buy and Sell Stock IV (DP)](http://www.programcreek.com/2014/03/leetcode-best-time-to-buy-and-sell-stock-iv-java/)  
[37) Longest Common Prefix](http://www.programcreek.com/2014/02/leetcode-longest-common-prefix-java/) [Google]  
[38) Largest Number](http://www.programcreek.com/2014/02/leetcode-largest-number-java/)  
[39) Combinations (DFS)](http://www.programcreek.com/2014/03/leetcode-combinations-java/)  
[40) Compare Version Numbers](http://www.programcreek.com/2014/03/leetcode-compare-version-numbers-java/)  
[41) Gas Station](http://www.programcreek.com/2014/03/leetcode-gas-station-java/)  
[42) Candy](http://www.programcreek.com/2014/03/leetcode-candy-java/) [Google]  
[43) Jump Game](http://www.programcreek.com/2014/03/leetcode-jump-game-java/)  
[44) Pascal's Triangle](http://www.programcreek.com/2014/03/leetcode-pascals-triangle-java/)  
[45) Container With Most Water](http://www.programcreek.com/2014/03/leetcode-container-with-most-water-java/)  
[46) Count and Say](http://www.programcreek.com/2014/03/leetcode-count-and-say-java/)  
[47) Repeated DNA Sequences](http://www.programcreek.com/2014/03/leetcode-repeated-dna-sequences-java/)  
[48) House Robber](http://www.programcreek.com/2014/03/leetcode-house-robber-java/)   
[49) Dungeon Game (DP)](http://www.programcreek.com/2014/03/leetcode-dungeon-game-java/)

**2. Linked List**

The implementation of a linked list is pretty simple in Java. Each node has a value and a link to next node.

|  |
| --- |
| **class** Node {  **int** val;  Node next;    Node(**int** x) {  val = x;  next = **null**;  }  } |

Two popular applications of linked list are stack and queue.

Stack

|  |
| --- |
| **class** Stack{  Node top;    **public** Node peek(){  **if**(top != **null**){  **return** top;  }    **return** **null**;  }    **public** Node pop(){  **if**(top == **null**){  **return** **null**;  }**else**{  Node temp = **new** Node(top.val);  top = top.next;  **return** temp;  }  }    **public** **void** push(Node n){  **if**(n != **null**){  n.next = top;  top = n;  }  }  } |

Queue

|  |
| --- |
| **class** Queue{  Node first, last;    **public** **void** enqueue(Node n){  **if**(first == **null**){  first = n;  last = first;  }**else**{  last.next = n;  last = n;  }  }    **public** Node dequeue(){  **if**(first == **null**){  **return** **null**;  }**else**{  Node temp = **new** Node(first.val);  first = first.next;  **return** temp;  }  }  } |

It is worth to mention that Java standard library already contains a class called "[Stack](http://docs.oracle.com/javase/7/docs/api/java/util/Stack.html)", and [LinkedList](http://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html) can be used as a Queue (add() and remove()). (LinkedList implements the Queue interface.) If you need a stack or queue to solve problems during your interview, you can directly use them.

Classic Problems:  
[1) Add Two Numbers](http://www.programcreek.com/2012/12/add-two-numbers/)  
[2) Reorder List](http://www.programcreek.com/2013/12/in-place-reorder-a-singly-linked-list-in-java/)  
[3) Linked List Cycle](http://www.programcreek.com/2012/12/leetcode-linked-list-cycle/)  
[4) Copy List with Random Pointer](http://www.programcreek.com/2012/12/leetcode-copy-list-with-random-pointer/)  
[5) Merge Two Sorted Lists](http://www.programcreek.com/2012/12/leetcode-merge-two-sorted-lists-java/)  
[6) Merge k Sorted Lists \*](http://www.programcreek.com/2013/02/leetcode-merge-k-sorted-lists-java/)  
[7) Remove Duplicates from Sorted List](http://www.programcreek.com/2013/01/leetcode-remove-duplicates-from-sorted-list/)  
[8) Partition List](http://www.programcreek.com/2013/02/leetcode-partition-list-java/)  
[9) LRU Cache](http://www.programcreek.com/2013/03/leetcode-lru-cache-java/)  
[10) Intersection of Two Linked Lists](http://www.programcreek.com/2014/02/leetcode-intersection-of-two-linked-lists-java/)

**3. Tree & Heap**

A tree normally refers to a binary tree. Each node contains a left node and right node like the following:

|  |
| --- |
| **class** TreeNode{  **int** value;  TreeNode left;  TreeNode right;  } |

Here are some concepts related with trees:

1. *Binary Search Tree*: for all nodes, left children <= current node <= right children
2. *Balanced vs. Unbalanced*: In a balanced tree, the depth of the left and right subtrees of every node differ by 1 or less.
3. *Full Binary Tree*: every node other than the leaves has two children.
4. *Perfect Binary Tree*: a full binary tree in which all leaves are at the same depth or same level, and in which every parent has two children.
5. *Complete Binary Tree*: a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible

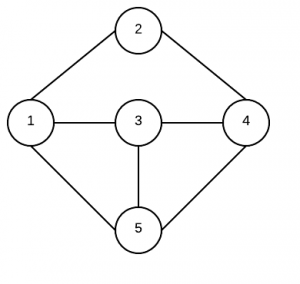
[Heap](http://en.wikipedia.org/wiki/Heap_%28data_structure%29) is a specialized tree-based data structure that satisfies the heap property. The time complexity of its operations are important (e.g., find-min, delete-min, insert, etc). In Java, [PriorityQueue](http://www.programcreek.com/2009/02/using-the-priorityqueue-class-example/) is important to know.

Classic problems:  
[1) Binary Tree Preorder Traversal](http://www.programcreek.com/2012/12/leetcode-solution-for-binary-tree-preorder-traversal-in-java/)   
[2) Binary Tree Inorder Traversal](http://www.programcreek.com/2012/12/leetcode-solution-of-binary-tree-inorder-traversal-in-java/) [Palantir]  
[3) Binary Tree Postorder Traversal](http://www.programcreek.com/2012/12/leetcode-solution-of-iterative-binary-tree-postorder-traversal-in-java/)  
[4) Word Ladder](http://www.programcreek.com/2012/12/leetcode-word-ladder/)  
[5) Validate Binary Search Tree](http://www.programcreek.com/2012/12/leetcode-validate-binary-search-tree-java/)  
[6) Flatten Binary Tree to Linked List](http://www.programcreek.com/2013/01/leetcode-flatten-binary-tree-to-linked-list/)  
[7) Path Sum](http://www.programcreek.com/2013/01/leetcode-path-sum/)  
[8) Construct Binary Tree from Inorder and Postorder Traversal](http://www.programcreek.com/2013/01/construct-binary-tree-from-inorder-and-postorder-traversal/)  
[9) Convert Sorted Array to Binary Search Tree](http://www.programcreek.com/2013/01/leetcode-convert-sorted-array-to-binary-search-tree-java/)  
[10) Convert Sorted List to Binary Search Tree](http://www.programcreek.com/2013/01/leetcode-convert-sorted-list-to-binary-search-tree-java/)  
[11) Minimum Depth of Binary Tree](http://www.programcreek.com/2013/02/leetcode-minimum-depth-of-binary-tree-java/)  
[12) Binary Tree Maximum Path Sum \*](http://www.programcreek.com/2013/02/leetcode-binary-tree-maximum-path-sum-java/)  
[13) Balanced Binary Tree](http://www.programcreek.com/2013/02/leetcode-balanced-binary-tree-java/)  
[14) Symmetric Tree](http://www.programcreek.com/2014/03/leetcode-symmetric-tree-java/)  
[15) Binary Search Tree Iterator](http://www.programcreek.com/2014/04/leetcode-binary-search-tree-iterator-java/)

**4. Graph**

Graph related questions mainly focus on depth first search and breath first search. Depth first search is straightforward, you can just loop through neighbors starting from the root node.

Below is a simple implementation of a graph and breath first search. The key is using a queue to store nodes.



1) Define a GraphNode

|  |
| --- |
| **class** GraphNode{  **int** val;  GraphNode next;  GraphNode[] neighbors;  **boolean** visited;    GraphNode(**int** x) {  val = x;  }    GraphNode(**int** x, GraphNode[] n){  val = x;  neighbors = n;  }    **public** String toString(){  **return** "value: "+ **this**.val;  }  } |

2) Define a Queue

|  |
| --- |
| **class** Queue{  GraphNode first, last;    **public** **void** enqueue(GraphNode n){  **if**(first == **null**){  first = n;  last = first;  }**else**{  last.next = n;  last = n;  }  }    **public** GraphNode dequeue(){  **if**(first == **null**){  **return** **null**;  }**else**{  GraphNode temp = **new** GraphNode(first.val, first.neighbors);  first = first.next;  **return** temp;  }  }  } |

3) Breath First Search uses a Queue

|  |
| --- |
| **public** **class** GraphTest {    **public** **static** **void** main(String[] args) {  GraphNode n1 = **new** GraphNode(1);  GraphNode n2 = **new** GraphNode(2);  GraphNode n3 = **new** GraphNode(3);  GraphNode n4 = **new** GraphNode(4);  GraphNode n5 = **new** GraphNode(5);    n1.neighbors = **new** GraphNode[]{n2,n3,n5};  n2.neighbors = **new** GraphNode[]{n1,n4};  n3.neighbors = **new** GraphNode[]{n1,n4,n5};  n4.neighbors = **new** GraphNode[]{n2,n3,n5};  n5.neighbors = **new** GraphNode[]{n1,n3,n4};    breathFirstSearch(n1, 5);  }    **public** **static** **void** breathFirstSearch(GraphNode root, **int** x){  **if**(root.val == x)  System.out.println("find in root");    Queue queue = **new** Queue();  root.visited = **true**;  queue.enqueue(root);    **while**(queue.first != **null**){  GraphNode c = (GraphNode) queue.dequeue();  **for**(GraphNode n: c.neighbors){    **if**(!n.visited){  System.out.print(n + " ");  n.visited = **true**;  **if**(n.val == x)  System.out.println("Find "+n);  queue.enqueue(n);  }  }  }  }  } |

Output:

value: 2 value: 3 value: 5 Find value: 5  
value: 4

Classic Problems:  
[1) Clone Graph](http://www.programcreek.com/2012/12/leetcode-clone-graph-java/)

**5. Sorting**

Time complexity of different sorting algorithms. You can go to wiki to see basic idea of them.

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Average Time | Worst Time | Space |
| Bubble sort | n^2 | n^2 | 1 |
| Selection sort | n^2 | n^2 | 1 |
| Insertion sort | n^2 | n^2 |  |
| Quick sort | n log(n) | n^2 |  |
| Merge sort | n log(n) | n log(n) | depends |

\* BinSort, Radix Sort and CountSort use different set of assumptions than the rest, and so they are not "general" sorting methods. (Thanks to Fidel for pointing this out)

Here are some implementations/demos, and in addition, you may want to check out how [Java developers sort in practice](http://www.programcreek.com/2014/03/how-developers-sort-in-java/).  
[1) Mergesort](http://www.programcreek.com/2012/11/leetcode-solution-merge-sort-linkedlist-in-java/)  
[2) Quicksort](http://www.programcreek.com/2012/11/quicksort-array-in-java/)  
[3) InsertionSort](http://www.programcreek.com/2012/11/leetcode-solution-sort-a-linked-list-using-insertion-sort-in-java/).  
[4) Maximum Gap (Bucket Sort)](http://www.programcreek.com/2014/03/leetcode-maximum-gap-java/)

**6. Recursion vs. Iteration**

Recursion should be a built-in thought for programmers. It can be demonstrated by a simple example.

Question:

there are n stairs, each time one can climb 1 or 2. How many different ways to climb the stairs?

*Step 1: Finding the relationship before n and n-1.*

To get n, there are only two ways, one 1-stair from n-1 or 2-stairs from n-2. If f(n) is the number of ways to climb to n, then f(n) = f(n-1) + f(n-2)

*Step 2: Make sure the start condition is correct.*

f(0) = 0;  
f(1) = 1;

|  |
| --- |
| **public** **static** **int** f(**int** n){  **if**(n <= 2) **return** n;  **int** x = f(n-1) + f(n-2);  **return** x;  } |

The time complexity of the recursive method is exponential to n. There are a lot of redundant computations.

f(5)  
f(4) + f(3)  
f(3) + f(2) + f(2) + f(1)  
f(2) + f(1) + f(2) + f(2) + f(1)

It should be straightforward to convert the recursion to iteration.

|  |
| --- |
| **public** **static** **int** f(**int** n) {    **if** (n <= 2){  **return** n;  }    **int** first = 1, second = 2;  **int** third = 0;    **for** (**int** i = 3; i <= n; i++) {  third = first + second;  first = second;  second = third;  }    **return** third;  } |

For this example, iteration takes less time. You may also want to check out [Recursion vs Iteration](http://www.programcreek.com/2012/10/iteration-vs-recursion-in-java/).

**7. Dynamic Programming**

Dynamic programming is a technique for solving problems with the following properties:

1. An instance is solved using the solutions for smaller instances.
2. The solution for a smaller instance might be needed multiple times.
3. The solutions to smaller instances are stored in a table, so that each smaller instance is solved only once.
4. Additional space is used to save time.

The problem of climbing steps perfectly fit those 4 properties. Therefore, it can be solve by using dynamic programming.

|  |
| --- |
| **public** **static** **int**[] A = **new** **int**[100];    **public** **static** **int** f3(**int** n) {  **if** (n <= 2)  A[n]= n;    **if**(A[n] > 0)  **return** A[n];  **else**  A[n] = f3(n-1) + f3(n-2);*//store results so only calculate once!*  **return** A[n];  } |

Classic problems:  
[1) Edit Distance](http://www.programcreek.com/2013/12/edit-distance-in-java/)  
[2) Longest Palindromic Substring](http://www.programcreek.com/2013/12/leetcode-solution-of-longest-palindromic-substring-java/)  
[3) Word Break](http://www.programcreek.com/2012/12/leetcode-solution-word-break/)  
[3) Word Break II](http://www.programcreek.com/2014/03/leetcode-word-break-ii-java/)  
[4) Maximum Subarray](http://www.programcreek.com/2013/02/leetcode-maximum-subarray-java/)  
[4) Maximum Product Subarray](http://www.programcreek.com/2014/03/leetcode-maximum-product-subarray-java/)  
[5) Palindrome Partitioning](http://www.programcreek.com/2013/03/leetcode-palindrome-partitioning-java/)  
[6) Candy](http://www.programcreek.com/2014/03/leetcode-candy-java/) [Google]  
[7) Jump Game](http://www.programcreek.com/2014/03/leetcode-jump-game-java/)  
[8) Best Time to Buy and Sell Stock III (DP)](http://www.programcreek.com/2014/02/leetcode-best-time-to-buy-and-sell-stock-iii-java/)  
[8) Best Time to Buy and Sell Stock IV (DP)](http://www.programcreek.com/2014/03/leetcode-best-time-to-buy-and-sell-stock-iv-java/)  
[9) Dungeon Game (DP)](http://www.programcreek.com/2014/03/leetcode-dungeon-game-java/)

**8. Bit Manipulation**

Bit operators:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OR (|) | AND (&) | XOR (^) | Left Shift (<<) | Right Shift (>>) | Not (~) |
| 1|0=1 | 1&0=0 | 1^0=1 | 0010<<2=1000 | 1100>>2=0011 | ~1=0 |

Get bit i for a give number n. (i count from 0 and starts from right)

|  |
| --- |
| **public** **static** **boolean** getBit(**int** num, **int** i){  **int** result = num & (1<<i);    **if**(result == 0){  **return** **false**;  }**else**{  **return** **true**;  }  } |

For example, get second bit of number 10.

i=1, n=10  
1<<1= 10  
1010&10=10  
10 is not 0, so return true;

Classic Problems:  
[1) Single Number](http://www.programcreek.com/2012/12/leetcode-solution-of-single-number-in-java/)  
[1) Single Number II](http://www.programcreek.com/2014/03/leetcode-single-number-ii-java/)  
[2) Maximum Binary Gap](http://www.programcreek.com/2013/02/twitter-codility-problem-max-binary-gap/)  
[3) Number of 1 Bits](http://www.programcreek.com/2014/03/leetcode-number-of-1-bits-java/)   
[4) Reverse Bits](http://www.programcreek.com/2014/03/leetcode-reverse-bits-java/)   
[5) Repeated DNA Sequences](http://www.programcreek.com/2014/03/leetcode-repeated-dna-sequences-java/)

**9. Probability**

Solving probability related questions normally requires formatting the problem well. Here is just a simple example of such kind of problems.

There are 50 people in a room, what's the probability that two people have the same birthday? (Ignoring the fact of leap year, i.e., 365 day every year)

Very often calculating probability of something can be converted to calculate the opposite. In this example, we can calculate the probability that all people have unique birthdays. That is: 365/365 \* 364/365 \* 363/365 \* ... \* 365-n/365 \* ... \* 365-49/365. And the probability that at least two people have the same birthday would be 1 - this value.

|  |
| --- |
| **public** **static** **double** caculateProbability(**int** n){  **double** x = 1;    **for**(**int** i=0; i<n; i++){  x \*= (365.0-i)/365.0;  }    **double** pro = Math.round((1-x) \* 100);  **return** pro/100;  } |

calculateProbability(50) = 0.97

**10. Combinations and Permutations**

The difference between combination and permutation is whether order matters.

Example 1:

Given 5 numbers - 1, 2, 3, 4 and 5, print out different sequence of the 5 numbers. 4 can not be the third one, 3 and 5 can not be adjacent. How many different combinations?

Example 2:

Given 5 banaba, 4 pear, and 3 apple, assuming one kind of fruit are the same, how many different combinations?

Class Problems:  
[1) Permutations](http://www.programcreek.com/2013/02/leetcode-permutations-java/)  
[2) Permutations II](http://www.programcreek.com/2013/02/leetcode-permutations-ii-java/)   
[3) Permutation Sequence](http://www.programcreek.com/2013/02/leetcode-permutation-sequence-java/)  
[4) Generate Parentheses](http://www.programcreek.com/2014/01/leetcode-generate-parentheses-java/)

**11. Math**

Solving math problems usually require us to get some observations and form rules:

[1) Reverse Integer](http://www.programcreek.com/2012/12/leetcode-reverse-integer/)  
[2) Palindrome Number](http://www.programcreek.com/2013/02/leetcode-palindrome-number-java/)  
[3) Pow(x,n)](http://www.programcreek.com/2012/12/leetcode-powx-n/)  
[4) Subsets](http://www.programcreek.com/2013/01/leetcode-subsets-java/)  
[5) Subsets II](http://www.programcreek.com/2013/01/leetcode-subsets-ii-java/)  
[6) Fraction to Recurring Decimal](http://www.programcreek.com/2014/03/leetcode-fraction-to-recurring-decimal-java/) [Google]  
[7) Excel Sheet Column Number](http://www.programcreek.com/2014/03/leetcode-excel-sheet-column-number-java/)  
[8) Excel Sheet Column Title](http://www.programcreek.com/2014/03/leetcode-excel-sheet-column-title-java/)   
[9) Factorial Trailing Zeroes](http://www.programcreek.com/2014/04/leetcode-factorial-trailing-zeroes-java/)

**Additional Resources**  
1. [Share your code to Github/BitBucket](http://www.programcreek.com/2013/02/how-to-share-your-eclipse-projects-to-github/)

**You may also like ...**

1. [LeetCode – Binary Search Tree Iterator (Java)](http://www.programcreek.com/2014/04/leetcode-binary-search-tree-iterator-java/)
2. [How to answer coding questions for your interview?](http://www.programcreek.com/2013/02/how-to-answer-coding-questions-for-your-interview/)
3. [面试10大算法汇总＋常见题目解答](http://www.programcreek.com/2012/12/%e9%9d%a2%e8%af%9510%e5%a4%a7%e7%ae%97%e6%b3%95%e6%b1%87%e6%80%bb%ef%bc%8b%e5%b8%b8%e8%a7%81%e9%a2%98%e7%9b%ae%e8%a7%a3%e7%ad%94/)
4. [Leetcode Solution of Iterative Binary Tree Postorder Traversal in Java](http://www.programcreek.com/2012/12/leetcode-solution-of-iterative-binary-tree-postorder-traversal-in-java/)