Basic Core Programs

1. **User Input and Replace String Template “Hello <<UserName>>, How are you?”**
    a. **I/P** -> Take User Name as Input. *Ensure UserName has min 3 char*
    b. **Logic** -> Replace <<UserName>> with the proper name
    c. **O/P** -> Print the String with User Name
2. **Flip Coin and print percentage of Heads and Tails**
    a. I/P -> The number of times to Flip Coin. *Ensure it is positive integer*.
    b. Logic -> Use Random Function to get value between 0 and 1. If < 0.5 then tails or heads
    c. O/P -> Percentage of Head vs Tails
3. **Leap Year**
    a. I/P -> Year, ensure it is a 4 digit number.
    b. Logic -> Determine if it is a Leap Year.
    c. O/P -> Print the year is a Leap Year or not.
4. **Power of 2**
    a. Desc -> This program takes a command-line argument N and prints a table of the powers of 2 that are less than or equal to 2^N.
    b. I/P -> The Power Value N. *Only works if 0 <= N < 31 since 2^31 overflows an int*
    c. Logic -> repeat until i equals N.
    d. O/P -> Print the year is a Leap Year or not.
5. **Harmonic Number**
    a. Desc -> Prints the Nth harmonic number: 1/1 + 1/2 + ... + 1/N (http://users.encs.concordia.ca/~chvatal/notes/harmonic.html).
    b. I/P -> The Harmonic Value N. *Ensure N != 0*
    c. Logic -> compute 1/1 + 1/2 + 1/3 + ... + 1/N
    d. O/P -> Print the Nth Harmonic Value.
6. **Factors**
    a. Desc -> Computes the prime factorization of N using brute force.
    b. I/P -> Number to find the prime factors
    c. Logic -> Traverse till i*i <= N instead of i <= N for efficiency.
    d. O/P -> Print the prime factors of number N.

Functional Programs

1. **2D Array**

    a. Desc -> A library for reading in 2D arrays of integers, doubles, or booleans from standard input and printing them out to standard output.
    b. I/P -> M rows, N Cols, and M * N inputs for 2D Array. Use Java Scanner Class
    c. Logic -> create 2 dimensional array in memory to read in M rows and N cols
    d. O/P -> Print function to print 2 Dimensional Array. In Java use PrintWriter with OutputStreamWriter to print the output to the screen.

2. **Sum of three Integer adds to ZERO**

    a. Desc -> A program with cubic running time. Read in N integers and counts the number of triples that sum to exactly 0.
    b. I/P -> N number of integer, and N integer input array
    c. Logic -> Find distinct triples (i, j, k) such that a[i] + a[j] + a[k] = 0
    d. O/P -> One Output is number of distinct triplets as well as the second output is to print the distinct triplets.

3. Write a program ***Distance.java*** that takes two integer command-line arguments x and y and prints the Euclidean distance from the point (x, y) to the origin (0, 0). The formulae to calculate distance = sqrt(x*x + y*y). Use Math.power function

4. Write a program ***Quadratic.java*** to find the roots of the equation a*x*x + b*x + c. Since the equation is x*x, hence there are 2 roots. The 2 roots of the equation can be found using a formula (Note: Take a, b and c as input values)

    delta = b*b - 4*a*c

    Root 1 of x = (-b + sqrt(delta))/(2*a)

    Root 2 of x = (-b - sqrt(delta))/(2*a)

5. Write a program ***WindChill.java*** that takes two double command-line arguments t and v and prints the wind chill. Use Math.pow(a, b) to compute ab. Given the temperature t (in Fahrenheit) and the wind speed v (in miles per hour), the National Weather Service defines the effective temperature (the wind chill) to be:

    $$w = 35.74 + 0.6215\,t + (0.4275\,t - 35.75)\,v^{0.16}$$

    Note: the formula is not valid if t is larger than 50 in absolute value or if v is larger than 120 or less than 3 (you may assume that the values you get are in that range).

Logical Programs

1. **Gambler**
    a. Desc -> Simulates a gambler who start with $stake and place fair $1 bets until he/she goes broke (i.e. has no money) or reach $goal. Keeps track of the number of times he/she wins and the number of bets he/she makes. Run the experiment N times, averages the results, and prints them out.
    b. I/P -> $Stake, $Goal and Number of times
    c. Logic -> Play till the gambler is broke or has won
    d. O/P -> Print Number of Wins and Percentage of Win and Loss.

2. **Coupon Numbers**
    a. Desc -> Given N distinct Coupon Numbers, how many random numbers do you need to generate distinct coupon number? This program simulates this random process.
    b. I/P -> N Distinct Coupon Number
    c. Logic -> repeatedly choose a random number and check whether it's a new one.
    d. O/P -> total random number needed to have all distinct numbers.
    e. Functions => Write Class Static Functions to generate random number and to process distinct coupons.

3. **Simulate Stopwatch Program**
    a. Desc -> Write a Stopwatch Program for measuring the time that elapses between the start and end clicks
    b. I/P -> Start the Stopwatch and End the Stopwatch
    c. Logic -> Measure the elapsed time between start and end
    d. O/P -> Print the elapsed time.

4. **Cross Game or Tic-Tac-Toe Game** 
    a. Desc -> Write a Program to play a Cross Game or Tic-Tac-Toe Game. Player 1 is the Computer and the Player 2 is the user. Player 1 take Random Cell that is the Column and Row.
    b. I/P -> Take User Input for the Cell i.e. Col and Row to Mark the ‘X’
    c. Logic -> The User or the Computer can only take the unoccupied cell. The Game is played till either wins or till draw...
    d. O/P -> Print the Col and the Cell after every step.
    e. Hint -> The Hints is provided in the Logic. Use Functions for the Logic…

Programs for JUnit Testing

1. **Find the Fewest Notes to be returned for Vending Machine**
    a. Desc -> There is 1, 2, 5, 10, 50, 100, 500 and 1000 Rs Notes which can be returned by Vending Machine. Write a Program to calculate the minimum number of Notes as well as the Notes to be returned by the Vending Machine as a Change
    b. I/P -> read the Change in Rs to be returned by the Vending Machine
    c. Logic -> Use Recursion and check for largest value of the Note to return change to get to minimum number of Notes.
    d. O/P -> Two Outputs - one the number of minimum Note needed to give the change and second list of Rs Notes that would given in the Change
2. To the Util Class add ***dayOfWeek*** static function that takes a date as input and prints the day of the week that date falls on. Your program should take three command-line arguments: m (month), d (day), and y (year). For m use 1 for January, 2 for February, and so forth. For output print 0 for Sunday, 1 for Monday, 2 for Tuesday, and so forth. Use the following formulas, for the Gregorian calendar (where / denotes integer division):

    $y_0 = y - (14 - m) / 12$

    $x = y_0 + y_0/4 - y_0/100 + y_0/400$

    $m_0 = m + 12 \times ((14 - m) / 12) - 2$

    $d_0 = (d + x + 31m_0 / 12) \bmod 7$
3. To the Util Class add ***temperaturConversion*** static function, given the temperature in fahrenheit as input outputs the temperature in Celsius or viceversa using the formula

    Celsius to Fahrenheit: (°C × 9/5) + 32 = °F

    Fahrenheit to Celsius: (°F − 32) x 5/9 = °C
4. Write a Util Static Function to calculate ***monthlyPayment*** that reads in three command-line arguments P, Y, and R and calculates the monthly payments you would have to make over Y years to pay off a P principal loan amount at R per cent interest compounded monthly. The formula is The formula is

```
                P r
payment =  ---------------,  where n = 12 * Y, r = R / (12 * 100)
           1  - (1 + r)^(-n)
```

5. Write a static function ***sqrt*** to compute the square root of a nonnegative number c given in the input using Newton's method:

   - initialize t = c

   - replace t with the average of c/t and t

   - repeat until desired accuracy reached using condition Math.abs(t - c/t) > epsilon*t where epsilon = 1e-15;

6. Write a static function ***toBinary*** that outputs the binary (base 2) representation of the decimal number typed as the input. It is based on decomposing the number into a sum of powers of 2. For example, the binary representation of 106 is 11010102, which is the same as saying that 106 = 64 + 32 + 8 + 2. Ensure necessary padding to represent 4 Byte String.

   To compute the binary representation of n, we consider the powers of 2 less than or equal to n in decreasing order to determine which belong in the binary decomposition (and therefore correspond to a 1 bit in the binary representation).

7. Write Binary.java to read an integer as an Input, convert to Binary using toBinary function and perform the following functions.

   i. Swap nibbles and find the new number.

   ii. Find the resultant number is the number is a power of 2.

   A nibble is a four-bit aggregation, or half an octet. There are two nibbles in a byte.

   Given a byte, swap the two nibbles in it. For example 100 is to be represented as 01100100 in a byte (or 8 bits). The two nibbles are (0110) and (0100). If we swap the two nibbles, we get 01000110 which is 70 in decimal.

Algorithm Programs

1. Write static functions to return all permutations of a String using iterative method and Recursion method. Check if the arrays returned by two string functions are equal.
2. **Binary Search the Word from Word List**
    a. Desc -> Read in a list of words from File. Then prompt the user to enter a word to search the list. The program reports if the search word is found in the list.
    b. I/P -> read in the list words comma separated from a File and then enter the word to be searched
    c. Logic -> Use Arrays to sort the word list and then do the binary search
    d. O/P -> Print the result if the word is found or not
3. **Insertion Sort**
    a. Desc -> Reads in strings and prints them in sorted order using insertion sort.
    b. I/P -> read in the list words
    c. Logic -> Use Insertion Sort to sort the words in the String array
    d. O/P -> Print the Sorted List
4. **Bubble Sort**
    a. Desc -> Reads in integers prints them in sorted order using Bubble Sort
    b. I/P -> read in the list ints
    c. O/P -> Print the Sorted List
5. **Merge Sort** - Write a program to do Merge Sort of list of Strings.
    a. Logic -> To Merge Sort an array, we divide it into two halves, sort the two halves independently, and then merge the results to sort the full array. To sort a[lo, hi), we use the following recursive strategy:
    b. Base case: If the subarray length is 0 or 1, it is already sorted.
    c. Reduction step: Otherwise, compute mid = lo + (hi - lo) / 2, recursively sort the two subarrays a[lo, mid) and a[mid, hi), and merge them to produce a sorted result.
6. **An Anagram Detection Example**
    a. Desc -> One string is an anagram of another if the second is simply a rearrangement of the first. For example, 'heart' and 'earth' are anagrams...
    b. I/P -> Take 2 Strings as Input such abcd and dcba and Check for Anagrams
    c. O/P -> The Two Strings are Anagram or not....
7. Take a range of 0 - 1000 Numbers and find the Prime numbers in that range.

8. Extend the above program to find the prime numbers that are Anagram and Palindrome

9. **Rewrite Use Generics for Search and Sort Algorithms**

10. **Question to find your number**

    a. Desc -> takes a command-line argument N, asks you to think of a number between 0 and N-1, where N = 2^n, and always guesses the answer with n questions.
    b. I/P -> the Number N and then recursively ask true/false if the number is between a high and low value
    c. Logic -> Use Binary Search to find the number
    d. O/P -> Print the intermediary number and the final answer

11. You have a long list of tasks that you need to do today. To accomplish task you need M minutes, and the deadline for this task is D . You need not complete a task at a stretch. You can complete a part of it, switch to another task, and then switch back.You've realized that it might not be possible to complete all the tasks by their deadline. So you decide to do them in such a manner that the maximum amount by which a task's completion time overshoots its deadline is minimized.

    Input Format - The first line contains the number of tasks, . Each of the next lines contains two integers,D and M .

    Output Format - Output *T* lines. The *ith* line contains the value of the maximum amount by which a task's completion time overshoots its deadline, when the first tasks on your list are scheduled optimally.

12. **Customize Message Demonstration using String Function and RegEx**

    a. Desc -> Read in the following message: Hello <<name>>, We have your full name as <<full name>> in our system. your contact number is 91-xxxxxxxxxx. Please,let us know in case of any clarification Thank you BridgeLabz 01/01/2016. Use Regex to replace name, full name, Mobile#, and Date with proper value.
    b. I/P -> read in the Message
    c. Logic -> Use Regex to do the following

        i. Replace <<name>> by first name of the user (assume you are the user)
        ii. replace <<full name>> by user full name.
        iii. replace any occurance of mobile number that should be in format 91-xxxxxxxxxx by your contact number.
        iv. replace any date in the format XX/XX/XXXX by current date.

    d. O/P -> Print the Modified Message.

Data Structure Programs

IMPORTANT NOTE - Use Generics to Solve all the Data Structure Programs

1. UnOrdered List

a. Desc -> Read the Text from a file, split it into words and arrange it as Linked List. Take a user input to search a Word in the List. If the Word is not found then add it to the list, and if it found then remove the word from the List. In the end save the list into a file
b. I/P -> Read from file the list of Words and take user input to search a Text
c. Logic -> Create a Unordered Linked List. The Basic Building Block is the Node Object. Each node object must hold at least two pieces of information. One ref to the data field and second the ref to the next node object.
d. O/P -> The List of Words to a File.

The Unordered List Abstract Data Type

The structure of an unordered list, as described above, is a collection of items where each item holds a relative position with respect to the others. Some possible unordered list operations are given below.

- `List()` creates a new list that is empty. It needs no parameters and returns an empty list.
- `add(item)` adds a new item to the list. It needs the item and returns nothing. Assume the item is not already in the list.
- `remove(item)` removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.
- `search(item)` searches for the item in the list. It needs the item and returns a boolean value.
- `isEmpty()` tests to see whether the list is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the list. It needs no parameters and returns an integer.
- `append(item)` adds a new item to the end of the list making it the last item in the collection. It needs the item and returns nothing. Assume the item is not already in the list.
- `index(item)` returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.
- `insert(pos,item)` adds a new item to the list at position pos. It needs the item and returns nothing. Assume the item is not already in the list and there are enough existing items to have position pos.
- `pop()` removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.
- `pop(pos)` removes and returns the item at position pos. It needs the position and returns the item. Assume the item is in the list.

2. Ordered List

a. Desc -> Read .a List of Numbers from a file and arrange it ascending Order in a Linked List. Take user input for a number, if found then pop the number out of the list else insert the number in appropriate position
b. I/P -> Read from file the list of Numbers and take user input for a new number
c. Logic -> Create a Ordered Linked List having Numbers in ascending order.
d. O/P -> The List of Numbers to a File.

The Ordered List Abstract Data Type

We will now consider a type of list known as an ordered list. For example, if the list of integers shown above were an ordered list (ascending order), then it could be written as 17, 26, 31, 54, 77, and 93. Since 17 is the smallest item, it occupies the first position in the list. Likewise, since 93 is the largest, it occupies the last position.

The structure of an ordered list is a collection of items where each item holds a relative position that is based upon some underlying characteristic of the item. The ordering is typically either ascending or descending and we assume that list items have a meaningful comparison operation that is already defined. Many of the ordered list operations are the same as those of the unordered list.

- `OrderedList()` creates a new ordered list that is empty. It needs no parameters and returns an empty list.
- `add(item)` adds a new item to the list making sure that the order is preserved. It needs the item and returns nothing. Assume the item is not already in the list.
- `remove(item)` removes the item from the list. It needs the item and modifies the list. Assume the item is present in the list.
- `search(item)` searches for the item in the list. It needs the item and returns a boolean value.
- `isEmpty()` tests to see whether the list is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the list. It needs no parameters and returns an integer.
- `index(item)` returns the position of item in the list. It needs the item and returns the index. Assume the item is in the list.
- `pop()` removes and returns the last item in the list. It needs nothing and returns an item. Assume the list has at least one item.
- `pop(pos)` removes and returns the item at position pos. It needs the position and returns the item. Assume the item is in the list.

3. Simple Balanced Parentheses

a. Desc -> Take an Arithmetic Expression such as (5+6)∗(7+8)/(4+3)(5+6)∗(7+8)/(4+3) where parentheses are used to order the performance of operations. Ensure parentheses must appear in a balanced fashion.
b. I/P -> read in Arithmetic Expression such as (5+6)∗(7+8)/(4+3)(5+6)∗(7+8)/(4+3)
c. Logic -> Write a Stack Class to push open parenthesis “(“ and pop closed parenthesis “)”. At the End of the Expression if the Stack is Empty then the

Arithmetic Expression is Balanced. Stack Class Methods are Stack(), push(), pop(), peak(), isEmpty(), size()

d. O/P -> True or False to Show Arithmetic Expression is balanced or not.

The Stack Abstract Data Type

The stack abstract data type is defined by the following structure and operations. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the "top." Stacks are ordered LIFO. The stack operations are given below.

- `Stack()` creates a new stack that is empty. It needs no parameters and returns an empty stack.
- `push(item)` adds a new item to the top of the stack. It needs the item and returns nothing.
- `pop()` removes the top item from the stack. It needs no parameters and returns the item. The stack is modified.
- `peek()` returns the top item from the stack but does not remove it. It needs no parameters. The stack is not modified.
- `isEmpty()` tests to see whether the stack is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items on the stack. It needs no parameters and returns an integer.

4. Simulate Banking Cash Counter

a. Desc -> Create a Program which creates Banking Cash Counter where people come in to deposit Cash and withdraw Cash. Have an input panel to add people to Queue to either deposit or withdraw money and dequeue the people. Maintain the Cash Balance.

b. I/P -> Panel to add People to Queue to Deposit or Withdraw Money and dequeue

c. Logic -> Write a Queue Class to enqueue and dequeue people to either deposit or withdraw money and maintain the cash balance

d. O/P -> True or False to Show Arithmetic Expression is balanced or not.

The Queue Abstract Data Type

The queue abstract data type is defined by the following structure and operations. A queue is structured, as described above, as an ordered collection of items which are added at one end, called the "rear," and removed from the other end, called the "front." Queues maintain a FIFO ordering property. The queue operations are given below.

- `Queue()` creates a new queue that is empty. It needs no parameters and returns an empty queue.
- `enqueue(item)` adds a new item to the rear of the queue. It needs the item and returns nothing.
- `dequeue()` removes the front item from the queue. It needs no parameters and returns the item. The queue is modified.
- `isEmpty()` tests to see whether the queue is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the queue. It needs no parameters and returns an integer.

As an example, if we assume that `q` is a queue that has been created and is currently empty, then Table 1 shows the results of a sequence of queue operations. The queue contents are shown such that the front is on the right. 4 was the first item enqueued so it is the first item returned by dequeue.

5. Palindrome-Checker

a. Desc -> A palindrome is a string that reads the same forward and backward, for example, radar, toot, and madam. We would like to construct an algorithm to input a string of characters and check whether it is a palindrome.
b. I/P -> Take a String as an Input
c. Logic -> The solution to this problem will use a deque to store the characters of the string. We will process the string from left to right and add each character to the rear of the deque.
d. O/P -> True or False to Show if the String is Palindrome or not.

The Deque Abstract Data Type

The deque abstract data type is defined by the following structure and operations. A deque is structured, as described above, as an ordered collection of items where items are added and removed from either end, either front or rear. The deque operations are given below.

- `Deque()` creates a new deque that is empty. It needs no parameters and returns an empty deque.
- `addFront(item)` adds a new item to the front of the deque. It needs the item and returns nothing.
- `addRear(item)` adds a new item to the rear of the deque. It needs the item and returns nothing.
- `removeFront()` removes the front item from the deque. It needs no parameters and returns the item. The deque is modified.
- `removeRear()` removes the rear item from the deque. It needs no parameters and returns the item. The deque is modified.
- `isEmpty()` tests to see whether the deque is empty. It needs no parameters and returns a boolean value.
- `size()` returns the number of items in the deque. It needs no parameters and returns an integer.

6. Hashing Function to search a Number in a slot

a. Desc -> Create a Slot of 10 to store Chain of Numbers that belong to each Slot to efficiently search a number from a given set of number
b. I/P -> read a set of numbers from a file and take user input to search a number
c. Logic -> Firstly store the numbers in the Slot. Since there are 10 Numbers divide each number by 11 and the reminder put in the appropriate slot. Create a Chain for each Slot to avoid Collision. If a number searched is found then pop it or else push it. Use Map of Slot Numbers and Ordered LinkedList to solve the problem. In the Figure Below, you can see number 77/11 reminder is 0 hence sits in the 0 slot while 26/11 remainder is 4 hence sits in slot 4
d. O/P -> Save the numbers in a file

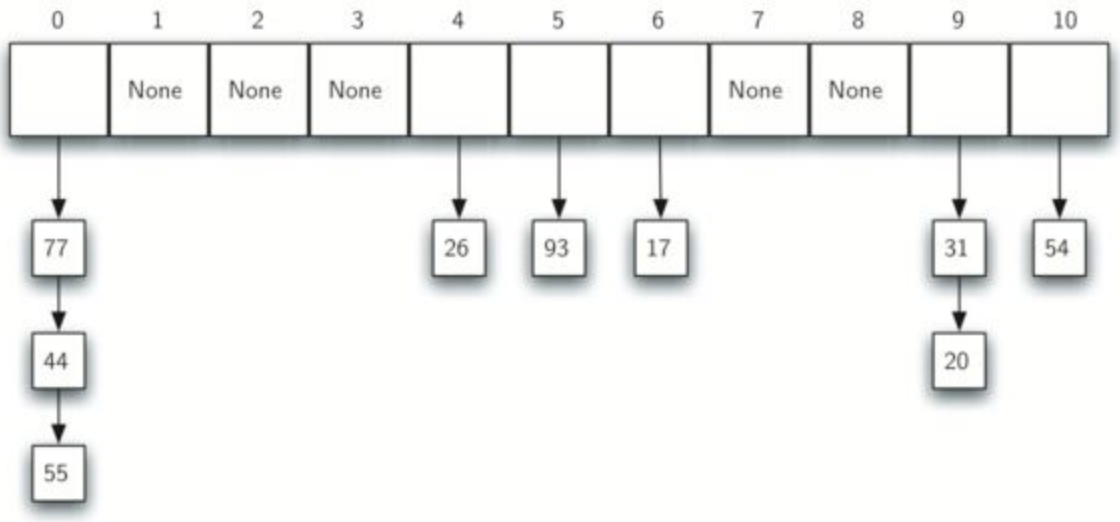


Figure 12: Collision Resolution with Chaining

7. **Number of Binary Search Tree**

   Solve the Problem in the following link
   https://www.hackerrank.com/challenges/number-of-binary-search-tree.

8. Take a range of 0 - 1000 Numbers and find the Prime numbers in that range. Store the prime numbers in a 2D Array, the first dimension represents the range 0-100, 100-200, and so on. While the second dimension represents the prime numbers in that range

9. Extend the Prime Number Program and store only the numbers in that range that are Anagrams. For e.g. 17 and 71 are both Prime and Anagrams in the 0 to 1000 range. Further store in a 2D Array the numbers that are Anagram and numbers that are not Anagram

10. Add the Prime Numbers that are Anagram in the Range of 0 - 1000 in a Stack using the Linked List and Print the Anagrams in the Reverse Order. Note no Collection Library can be used.

11. Add the Prime Numbers that are Anagram in the Range of 0 - 1000 in a Queue using the Linked List and Print the Anagrams from the Queue. Note no Collection Library can be used.

12. Write a program ***Calendar.java*** that takes the month and year as command-line arguments and prints the Calendar of the month. Store the Calendar in an 2D Array, the first dimension the week of the month and the second dimension stores the day of the week. Print the result as following.

```
% java Calendar 7 2005
 July 2005
 S  M  T  W Th  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

13. Create the Week Object having a list of WeekDay objects each storing the day (i.e S,M,T,W,Th,..) and the Date (1,2,3..) . The WeekDay objects are stored in a Queue implemented using Linked List. Further maintain also a Week Object in a Queue to finally display the Calendar

    Note - If a particular day has no date then the date is set as empty string and add it to queue.

14. Modify the above program to store the Queue in two Stacks. Stack here is also implemented using Linked List and not from Collection Library

Object Oriented Programs

1. **Address Book Problem.**

2. **JSON Inventory Data Management of Rice, Pulses and Wheats**

    a. Desc -> Create a JSON file having Inventory Details for Rice, Pulses and Wheats with properties name, weight, price per kg.
    b. Use Library : Java JSON Library, For IOS JSON Library use NSJSONSerialization for parsing the JSON.
    c. I/P -> read in JSON File
    d. Logic -> Get JSON Object in Java or NSDictionary in iOS. Create Inventory Object from JSON. Calculate the value for every Inventory.
    e. O/P -> Create the JSON from Inventory Object and output the JSON String

3. **Inventory Management Program**

    a. Desc -> Extend the above program to Create InventoryManager to manage the Inventory. The Inventory Manager will use InventoryFactory to create Inventory Object from JSON. The InventoryManager will call each Inventory Object in its list to calculate the Inventory Price and then call the Inventory Object to return the JSON String. The main program will be with InventoryManager
    b. I/P -> read in JSON File
    c. Logic -> Get JSON Object in Java or NSDictionary in iOS. Create Inventory Object from JSON. Calculate the value for every Inventory.
    d. O/P -> Create the JSON from Inventory Object and output the JSON String.

4. **Stock Account Management**
    a. Desc -> Write a program to read in Stock Names, Number of Share, Share Price. Print a Stock Report with total value of each Stock and the total value of Stock.
    b. I/P -> N number of Stocks, for Each Stock Read In the Share Name, Number of Share, and Share Price
    c. Logic -> Calculate the value of each stock and the total value
    d. O/P -> Print the Stock Report.
    e. Hint -> Create Stock and Stock Portfolio Class holding the list of Stocks read from the input file. Have functions in the Class to calculate the value of each stock and the value of total stocks
5. **Commercial data processing** - StockAccount.java implements a data type that might be used by a financial institution to keep track of customer information. The StockAccount class implements following methods

public class StockAccount		
	StockAccount(String filename)	*create a new account from file*
double	valueOf()	*total value of account dollars*
void	buy(int amount, String symbol)	*add shares of stock to account*
void	sell(int amount, String symbol)	*subtract shares of stock from account*
void	save(String filename)	*save account to file*
void	printReport()	*print a detailed report of stocks and values*

The StockAccount class also maintains a list of CompanyShares object which has Stock Symbol and Number of Shares as well as DateTime of the transaction. When buy or sell is initiated StockAccount checks if CompanyShares are available and accordingly update or create an Object.

6. Maintain the List of CompanyShares in a Linked List So new CompanyShares can be added or removed easily. Do not use any Collection Library to implement Linked List.
7. Further maintain the Stock Symbol Purchased or Sold in a Stack implemented using Linked List to indicate transactions done.
8. Further maintain DateTime of the transaction in a Queue implemented using Linked List to indicate when the transactions were done.

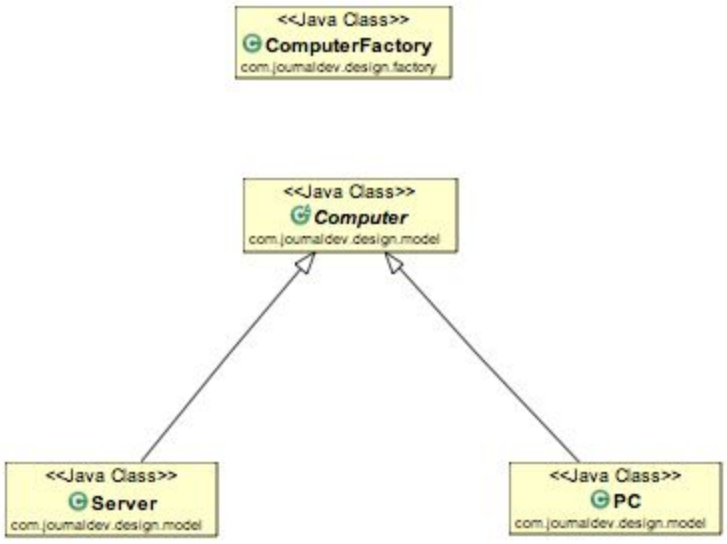
9. Write a Program ***DeckOfCards.java***, to initialize deck of cards having suit ("Clubs", "Diamonds", "Hearts", "Spades") & Rank ("2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"). Shuffle the cards using Random method and then distribute 9 Cards to 4 Players and Print the Cards the received by the 4 Players using 2D Array…
10. Extend the above program to create a Player Object having Deck of Cards, and having ability to Sort by Rank and maintain the cards in a Queue implemented using Linked List. Do not use any Collection Library. Further the Player are also arranged in Queue. Finally Print the Player and the Cards received by each Player.

11. **Clinique Management Programme.** This programme is used to manage a list of Doctors associated with the Clinique. This also manages the list of patients who use the clinique. It manages Doctors by Name, Id, Specialization and Availability (AM, PM or both). It manages Patients by Name, ID, Mobile Number and Age. The Program allows users to search Doctor by name, id, Specialization or Availability. Also the programs allows users to search patient by name, mobile number or id. The programs allows patients to take appointment with the doctor. A doctor at any availability time can see only 5 patients. If exceeded the user can take appointment for patient at different date or availability time. Print the Doctor Patient Report. Also show which Specialization is popular in the Clinique as well as which Doctor is popular. For .NET Engineers use the following
    a. ADO.NET Connection Pooling to maintain Doctor, Patient and Appointment Info in the Database
    b. Use Log4NET to Log Data
    c. Read Patient and Doctor Data from JSON File using File IO and latter with Firebase. Use Factory Pattern and Interface Approach to read Doctor and Patient information.

Design Pattern Programs

Creational Design Patterns

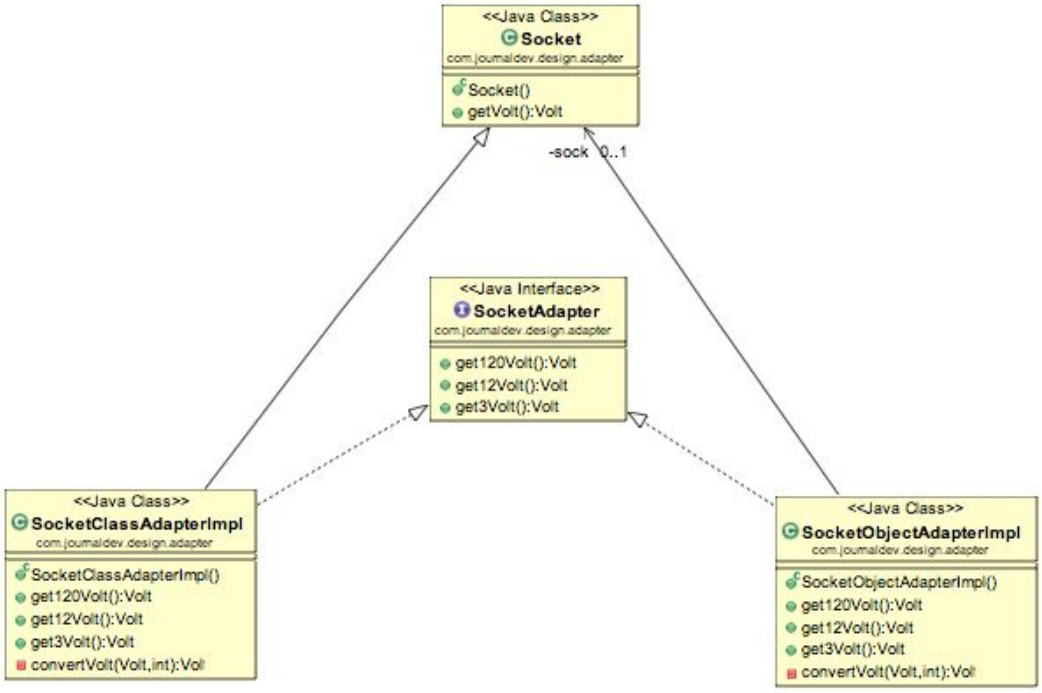
1. Refer **SIngleton Link** and practice the various SIngleton Approaches that are possible. This includes
   a. Eager initialization
   b. Static block initialization
   c. Lazy Initialization
   d. Thread Safe Singleton
   e. Bill Pugh Singleton Implementation
   f. Using Reflection to destroy Singleton Pattern
   g. Enum Singleton
   h. Serialization and Singleton
2. Use **Factory Pattern** to create a Computer Factory that can Produce PC, Laptop and Server Class Computers. As Shown in the Figure Below Create an Abstract Computer Class and PC, Laptop and Server inherit from Computer and ComputerFactory is able to create the corresponding Computer Object on request



3. **Prototype design pattern** is used when the Object creation is a costly affair and requires a lot of time and resources and you have a similar object already existing. Use Prototype Pattern as shown in the Link above to create multiple Employee Object

Structural Design Patterns

1. **Adapter design pattern** is one of the structural design pattern and its used so that two unrelated interfaces can work together. The object that joins these unrelated interface is called an Adapter. Use Adapter design pattern to solve mobile charger which adapts to a Mobile battery needs 3 volts to charge but the normal socket produces either 120V (US) or 240V (India). So the mobile charger works as an adapter between mobile charging socket and the wall socket.



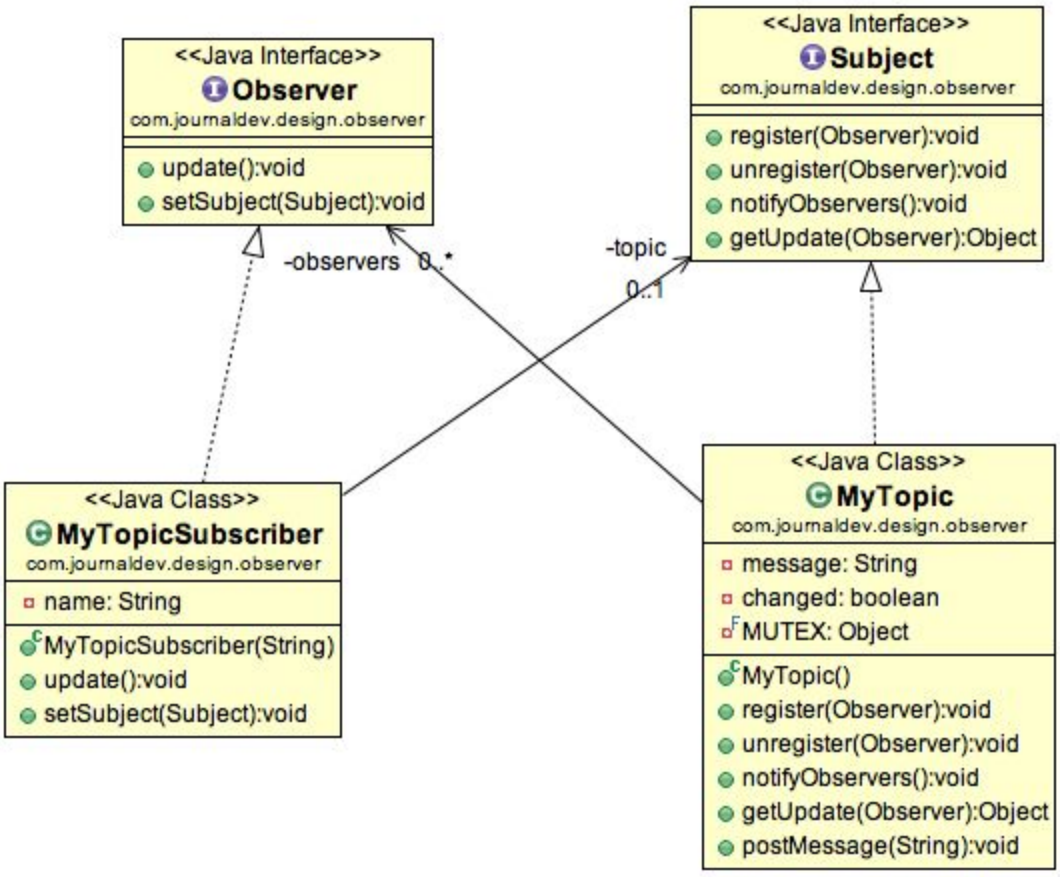
2. **Facade design pattern** is used to help client applications to easily interact with the system. In the Address Book Problem use Facade Pattern to read the Address Book from the File or from the Database

3. Proxy design pattern as the name suggests creates a Proxy Object to a real Object so as to provide controlled access to a functionality. Create a Command Executor Program that will execute certain system commands based on the user type is admin or otherwise. The Proxy design pattern link shows the same example.

Behavioral Design Patterns

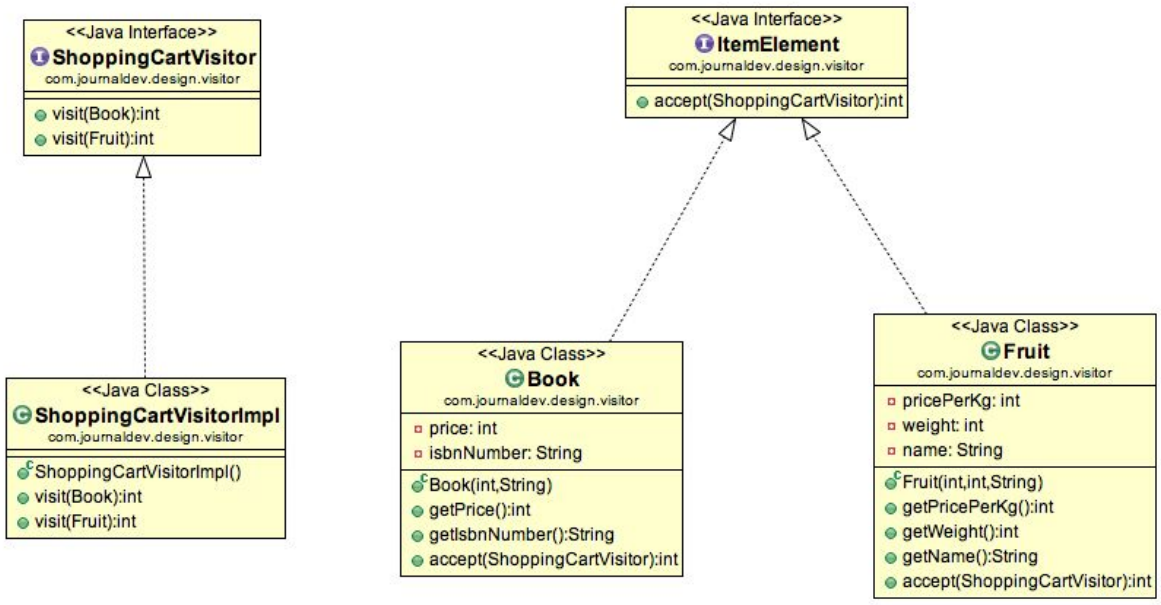
1. **Observer design pattern** is useful when you are interested in the state of an object and want to get notified whenever there is any change. In observer pattern, the object that watch on the state of another object are called Observer and the object that is being watched is called Subject.

   For observer pattern java program example, implement a simple topic and observers can register to this topic. Whenever any new message will be posted to the topic, all the registers observers will be notified and they can consume the message.



2. **Visitor pattern** is used when we have to perform an operation on a group of similar kind of Objects. With the help of visitor pattern, we can move the operational logic from the objects to another class.

   For example, think of a Shopping cart where we can add different type of items (Elements). When we click on checkout button, it calculates the total amount to be paid. Now we can have the calculation logic in item classes or we can move out this logic to another class using visitor pattern. Let's implement this in our example of visitor pattern.



3. **Mediator Design Pattern** is very helpful in an enterprise application where multiple objects are interacting with each other. If the objects interact with each other directly, the system components are tightly-coupled with each other that makes higher maintainability cost and not hard to extend. Mediator pattern focuses on provide a mediator between objects for communication and help in implementing loose-coupling between objects.

   Allows loose coupling by encapsulating the way disparate sets of objects interact and communicate with each other.