# Assignment - 1

## Set- 2

1. How to implement precedence rules and associativity in Java language? Give an example.

A. In Java when an expression is evaluated, there may be more than one operators involved in an expression When more than one operator has to be evaluated in an expression Java interpreter has to decide which operator should be evaluated first. Java makes this decision on the basis of the precedence and associativity of the operators.

**Java operators Precedence and Associativity:-**

Java operators have two properties. They are

1. Precedence
2. Associativity

1. **Precedence:-** Precedence is the priority order of an operator, if there are two or more operators in an expression then the operator of highest priority will be executed first then higher, and then high.

   Example:- `1+2*3`

   In the above expression multiplication (*) operator will be processed first and then addition. It's because multiplication has higher priority or precedence than addition.

2. Associativity:- Associativity tells the direction of execution of operators that can be either left to right or right to left.

Example:- a = b = c = 8

In the above expression the assignment operator is executed from right to left that means c will be assigned by 8, then be will be assigned by c and finally a will be assigned by b.

| Operator | Name | Associativity |
|---|---|---|
| ( ) | Parantheses | Left to right |
| ( ) | Function call | left to right |
| [ ] | Array Subscript | Left to right |
| . | Object member access | left to right |
| ++ | Post increment | left to right |
| -- | Post decrement | left to right |
| ++ | Pre increment | Right to left |
| -- | Pre decrement | Right to left |
| + | unary plus | Right to left |
| - | unary minus | Right to left |
| ! | unary logical negation | Right to left |
| (type) | unary casting | Right to left |
| new | Creating object | Right to left |
| * | Multiplication | left to right |
| / | Division | left to right |
| % | Remainder | left to right |
| + | Addition | left to right |
| - | Subtraction | left to right |

| Operator | name | Associativity |
|---|---|---|
| <<<br>>><br>>>> | left shift<br>Right shift with sign extension<br>left shift with zero extension. | left to right<br>left to right<br>left to right |
| <<br><=<br>><br>>= | less than<br>less than or equal to<br>Greater than<br>Greater than or equal to | left to right<br>left to right<br>left to right<br>left to right |
| instance of | checking object type | left to right |
| ==<br>!= | Equal comparison<br>Not equal | left to right<br>left to right |
| & | (unconditional AND) | left to right |
| ^ | (Exclusive OR) | left to right |
| \| | (unconditional OR) | left to right |
| && | Conditional AND | left to right |
| \|\| | Conditional OR | left to right |
| ?: | Ternary condition | Right to left |
| =<br>+=<br>-=<br>*=<br>/=<br>%= | Assignment<br>Addition Assignment<br>Subtraction Assignment<br>Multiplication Assignment<br>Division Assignment<br>Remainder Assignment | Right to left<br>Right to left<br>Right to left<br>Right to left<br>Right to left<br>Right to left |
| | | |

3. Define a class Electric Bill with the following specifications

Class: Electric Bill

Instance Variable / data members:

String n - to store the name of the customer

int units - to store the number of units consumed

double bill - to store the amount to paid

Member methods:

Void accept() - to accept the name of the customer and number of units consumed

Void calculate() - to calculate the bill as per the following tariff:

Number of units - Rate per unit

First 100 units - Rs 2.00

Next 200 units - Rs 3.00

Above 300 units - Rs 5.00

A surcharge of 2.5% charged if the number of units consumed is above 300 units.

Void print() - To print the details as follows.

Name of the customer....

Number of units consumed.....

Bill amount.....

Write a main method to create an object of the class and call the above member methods.

```
import java.io.*;
class ElectricBill{
    String n;
    int units;
    double bill;
Void accept()throws IOException{
    BufferedReader reader: new BufferedReader(new Input
                                StreamReader(System.in));
     System.out.print("Name of the customer :");
     n=reader.readline();
     System.out.print("Number of units consumed =");
     String un = reader.readLine();
     units = Integer.parseInt(un);
}
Void calculate(){
    if (units <=100)
        bill = 2.00 * units;
    if (units <=300)
        bill = 3.00 * units;
     if (units > 300)
        bill = 5.00 * units;
}
```

```
Void print() {
        System.out.println("Name of the customer:"+n);
        System.out.println("Number of units consumed:"+units);
        System.out.println("Bill amount:"+bill);
}
public static void main(String args[])throws IO Exception{
        ElectricBill eb = new ElectricBill();
        eb.accept();
        eb.calculate();
}
}
```

4. Design a class to overload a function check() as follows:

i, Void check (String str, char ch) – to find and print the frequency of a character in a string.

Example :-

Input – output

Str = "success" number of s present is = 3

ch = 's'

ii, Void check (String s1) – to display only the vowels from string s1, after converting it to lower case

Example :

Input:

S1 = "Computer" output : o u e

19 19BQ1A05H8

```
class Character Vowel {
    public void check(String str, char ch){
        int c=0, code, i,s;
        str = str.toLowerCase();
        int len = str.length()
        for (code = 97; code <122; code++){
            c=0;
        for  -for (i=0; i<len; i++){
                ch = str.charAt(i);
                s =(int) ch;
                if (s == code)
                    c=c+1;
                }
            ch = (char) code)
            if(c!=0)
            System.out.println("Frequency of "+ch+"is"+c);
        }
    }
    public void check (String si){
        int i;
        char ch=0; chr=0;
        for(i=0; i< si.length(); i++){
            ch = si.charAt(i);
            if (Character.isUpperCase(ch))
            Chr = Character.toLowerCase(ch);
```

```
if ((s1.charAt(i)=='a' || (s1.charAt(i)=='e') ||
    (s1.charAt(i)=='o') || (s1.charAt(i)=='i') ||
    (s1.charAt(i)=='u'))
        System.out.println(s1.charAt(i));
    }
  }
}
```

2. Design a class that represents a bank account and construct the methods to

i, Assign initial values

ii, Deposit an amount

iii, Withdraw amount after checking balance

iv, Display the name and balance.

Do you need to use static keyword for the above bank account program? Explain.

19BQ1A05H8

```
public class SavingsAccount
{
    private double balance;
    private double interest;
    private String name;

    public SavingsAccount()
    {
       balance = 0;
       interest = 0;
       name = "Tina";
    }
    public SavingsAccount(double initialBalance, double initial
                                    Interest, String initialName)
    {
          balance = initialBalance;
          interest = initialInterest;
          name = initialName;
    }
    public void deposit(double amount)
    {
       balance = balance + amount);
    }
    public void withdraw(double amount)
    {
       balance = balance - amount;
    }
```

19BAIDOSHS

```
    public  void addInterest()
    {
       balance = balance + balance *interest;
    }
    public double getBalance()
    {
       return balance;
    }
    public String getName()
    {
        return name;
    }
}
```

We need not use static keyword for the above program. The most common example of a static method is the main() method. Methods declared as static can have the following restrictions

* They can directly call other static methods only
* They can access static data directly.

Resources:-

Introduction to Java Programming