1. a. JAVA Program to demonstrate Constructor Overloading and Method Overloading.

```
class Test1a
{
  int a;

  Test1a()
  {
    a=1;
   }

  Test1a(int i)
  {
    a=i;
   }

 void show()
   {
    System.out.println("\n Value of a : " +a);
   }

 void show(int m)
 {
  System.out.println("\n Value of argument passed : " +m);
 }
}

class Q1a
```

```
{
 public static void main(String args[])
 {
   Test1a ob1=new Test1a();
   Test1a ob2=new Test1a(10);


   ob1.show();
   ob2.show();
   ob2.show(33);


 }
}
```

1.b. JAVA Program to implement Inner class and demonstrate its Access Protections.

```
class Outer

{

        int x=100;
        int y=10;
        void test()

        {

                Inner ob1=new Inner();

                ob1.display();

        }

        class Inner

        {

          int z;
          Inner()

           {

              y=90;

              z=60;

           }

        void display()

                {
                        System.out.println("Display : x = " +x);

                System.out.println("Display : y =" +y);
                System.out.println("Display : z = " +z);

                }

        }//Inner class closes here
void show()
          {
              System.out.println("Display : x = " +x);

                   System.out.println("Display : y = " +y);
                 //   System.out.println("Display : z = " +z);
          }
 }//outer class closes here


 class Q1b
  {
          public static void main(String args[])
           {
                   Outer ob1=new Outer();
                   ob1.test();
                  ob1.show();
           }
  }
```

2. Write a program in Java for String handling which performs the following:

i) Checks the capacity of StringBuffer object.

ii) Reverse the contents of a string given on console and converts the resultant string in upper case.

iii) Reads a string from console and append it to the resultant string of ii.

i) public class StringBuf

```
{
   public static void main(String args[ ] )
   {
      StringBuffer buffer1 = new StringBuffer( ) ;
      StringBuffer buffer2 = new StringBuffer(50) ;
      StringBuffer buffer3 = new StringBuffer("hello") ;
      System.out.println("buffer1 capacity: " + buffer1.capacity());
      System.out.println("buffer2 capacity: " + buffer2.capacity());
      System.out.println("buffer3 capacity: " + buffer3.capacity());

      System.out.println("\nbuffer1 length: " + buffer1.length());
      System.out.println("buffer2 length: " + buffer2.length());
      System.out.println("buffer3 length: " + buffer3.length());
      buffer3.ensureCapacity(150);
      System.out.println("After modifying buffer3 capacity: " + buffer3.capacity());
   }
}
```

ii) And iii)

```
import java.util.Scanner;

class Two_a

{

  public static void main(String args[])

  {

    String original, reverse = "appnd”;
    Scanner in = new Scanner(System.in);

    System.out.println("Enter a string to reverse : ");

    original = in.nextLine();

    int length = original.length();

    for ( int i = length - 1 ; i >= 0 ; i-- )

     reverse = reverse + original.charAt(i);

     System.out.println("Reverse of entered string is: "+reverse);

     System.out.println("String in Upper case: "+reverse.toUpperCase());

     System.out.println("Enter a string to appends: ");

     appnd = in.nextLine();

     System.out.println("After Appending: "+reverse.toUpperCase().concat(appnd));
```

```
                  }

               }
```

3. a. Write a JAVA Program to implement Inheritance.

```
  class Parent
{
int x;
Parent(int a)
{
x=a;
}

  void displayx()
   {
    System.out.println("\n Value of i : " +x);
   }
 }



 class Child extends Parent
 {
   int y;
   Child(int a,int b)
  {
    super(a);
    y=b;
  }
 void displayxy()
 {
   System.out.println("\n Value of a : " +x);


   System.out.println("\n Value of b : " +y);
  }
}


class Inheritance
{
  public static void main(String args[])
  {
      Parent ob1=new Parent(10);
      Child  ob2=new Child(20,30);
      System.out.println("\n Contents of Parent or Super Class Object : ");
      ob1.displayx();
      System.out.println("\n Contents of Child Class Object : ");
      ob2.displayxy();
   }
}
```

3. b.   Multiple inheritance using interface

```
interface Shape{
void area();
} // end of interface



class Rectangle implements Shape{
double l,b;
Rectangle(double length, double breadth)
{
l=length;
b=breadth;
}
public void area(){
System.out.println("Area of Rectangle is : " + l*b);
}
}

class Triangle implements Shape
{
double b,h;
Triangle (double base, double height){
b= base;
h= height;
}

public void area()
{
System.out.println("Area of Triangle is : " + (b*h/2));
}
}



public class InterfaceDemo {
  public static void main(String args[]){
  Rectangle rect=new Rectangle(10,05);

rect.area();

Triangle tri=new Triangle(10,20);
tri.area();

  }// end of main
} //end of InterfaceDemo
```

OUTPUT:

```
Area of Rectangle is : 50.0
Area of Triangle is : 100.0
```

.

4. Write a JAVA program which has

   i. A Interface class for Stack Operations

   ii. A Class that implements the Stack Interface and creates a fixed length Stack.

   iii. A Class that implements the Stack Interface and creates a Dynamic length Stack.

   iv. A Class that uses both the above Stacks through Interface reference and does the Stack operations that demonstrates the runtime binding.

```
 import java.util.*;

 interface mystackinterface
 {
    void push(int a);
    int pop();
    boolean isempty();
 }



 class fixedstack implements mystackinterface
 {
int  top;
int st[];

fixedstack()
```

```
{
   top=-1;
   st=new int[10];
}




public boolean isempty()
{
   if (top==-1)
      return true;
   else
      return false;
 }


public void push(int a)
{
   if(top!=9)
   {
      st[++top] =a;
      System.out.println(a+" Added to FIXED LENGTH STACK");
   }
   else
      System.out.println("Fixed Length Stack full");
}
```

```
    public int pop()
    {
        return st[top--];
    }
}







class dynamictack implements mystackinterface
{
    int top; ArrayList st;

    dynamictack()
    {
        top=-1;
        st=new ArrayList();
    }


    public boolean isempty()
    {
        if (top==-1)
            return true;
        else
            return false;
```

```
     }


    public void push(int a)
    {
        top++;
        st.add(a);
    }


    public int pop()
    {
         Integer ob=(Integer)st.remove(top--);
         return ob.intValue();
    }
}
public class mystackimpl
{
    public static void main(String[] args)
    {
      mystackinterface fstk=new fixedstack();
      mystackinterface dstk=new dynamictack();

 for(int i=0;i<15;i++)
            fstk.push(i);

        for(int i=0;i<15;i++)
        if(!fstk.isempty())
            System.out.println("Top Element of Fixed Length Stack : "+fstk.pop());
         else
```

```
            System.out.println("FIXED LENGTH STACK IS EMPTY");

        for(int i=0;i<15;i++)
          {
            dstk.push(i);
            System.out.println(i+" Added to Dynamic LENGTH STACK");
          }

        for (int i = 0; i < 15; i++)
         if(!dstk.isempty())
            System.out.println("Top Element of Dynamic Length Stack : "+dstk.pop());
         else
            System.out.println("Dynamic LENGTH STACK IS EMPTY");
    }
}
```

Q.5: Shape package and implementations ………

Create a package, Shape separately for each……

```
package shape;

public class Circle {

    private double r, PI=3.14;
    public Circle(double radius)
    {
```

```
    r=radius;
    }
    public void area(){
    System.out.println("Area of circle ......."+(PI*r*r));
    }

}
```

```
package shape;
public class Square
{
    private double a;
    public Square(double side)
    {
    a=side;
    }
    public void area(){
    System.out.println("Area of Square ......."+(4*a));
    }

}
```

```
package shape;
public class TriangleOne
{
    private double b,h;
    public TriangleOne(double base, double height)
    {
```

```
    b=base; h=height;
    }
   public void area(){
    System.out.println("Area of Triangle ......."+(b*h / 2));
    }
}
```

```
import shape.*;
public class Lab7 {
    public static void main(String arg[]){
    Circle c=new Circle(20);
    c.area();
    Square s=new Square(15);
    s.area();
   TriangleOne obj = new TriangleOne(12,6);
   obj.area();
    }
}
```

Output:

```
Area of Circle is : 1256.0
Area of Square .......60.0
Area of Triangle .......36.0
```

6. Write a JAVA program which has

i. A Class called Account that creates account with 500Rs minimum balance, a deposit() method to deposit amount, a withdraw() method to withdraw amount and also throws LessBalanceException if an account holder tries to withdraw money which makes the balance become less than 500Rs.

ii. A Class called LessBalanceException which returns the statement that says withdraw amount (___Rs) is not valid.

iii. A Class which creates 2 accounts, both account deposit money and one account tries to withdraw more money which generates a LessBalanceException take appropriate action for the same

```
class Account
{
   int bal=500;
   void deposit(int amt)
   {
      bal+=amt;
   }

   void withdraw(int amt)
   {
      if ((bal-amt)<=500)
      {
         try
            {
                  throw new LessBalanceException(amt);
              }catch (LessBalanceException e1)
               {
               }
       }
      else
         bal-=amt;
   }
}
class LessBalanceException extends Exception
```

```
 {


     LessBalanceException(int amt)
     {
         System.out.println("Withdrawing of "+amt+"  not possible");
     }
 }



 public class Q3 {
     public static void main(String[] args)
     {
      Account ob1=new Account();

      Account ob2=new Account();

      ob1.deposit(200);

      ob2.deposit(200);

      ob2.withdraw(300);

      ob2.withdraw(100);

      System.out.println("Current Balance for ob1 = "+ob1.bal);

      System.out.println("Current Balance for ob2 = "+ob2.bal);

     }
}
```

7. Queue Operations using user defined exceptions

```
import java.util.Scanner;

class ExcQueue extends Exception
{
     ExcQueue(String s)
    {
        super(s);
     }
  }

class Queue
 {
     int front,rear;
     int q[ ]=new int[10];

     Queue()
    {
        rear=-1;
        front=-1;
      }

    void enqueue(int n) throws ExcQueue
    {
          if (rear==9) throw new ExcQueue("Queue is full");
```

```
            rear++;
            q[rear]=n;
          if (front==-1)   front=0;
    }

int dequeue() throws ExcQueue
{
       if (front==-1) throw new ExcQueue("Queue is empty");
       int temp=q[front];
        if (front==rear)
              front=rear=-1;
        else
               front++;
       return(temp);
     }
} //Class closes here


  class UseQueue
 {
       public static void main(String args[ ])
      {
            Queue a=new Queue();
            try
          {
              a.enqueue(5);
              a.enqueue(20);
           }
           catch (ExcQueue e)
          {
              System.out.println(e.getMessage());
           }



            try
            {
              System.out.println(a.dequeue());
              System.out.println(a.dequeue());
              System.out.println(a.dequeue());
              }
              catch(ExcQueue e)
            {
                System.out.println(e.getMessage());
            }
      }
}
```

8. Write a JAVA program using Synchronized Threads, which demonstrates Producer Consumer concept.

```
import java.lang.*;
import java.io.*;
import java.util.*;


class common

{

 boolean flag=false;

 String str;


 public synchronized void produce() throws Exception

  {

    if(flag==false)

       {

        Scanner  sc=new  Scanner(System.in);
        System.out.println("enter the string");
        str=sc.next();
        flag=true;
        notify();
      }
   else
        wait();
   }

  public synchronized void consume() throws Exception
    {
     if(flag==true)
      {
        System.out.println("the produced string by producer is : "+str);
        flag=false;
        notify();
      }
     else
            wait();
    }
}
```

```
class producer extends Thread
{
  common c;
  producer(common c)
 {
  this.c=c;
   }

  public void run()
    {
      try
          {
           c.produce();
          }catch(Exception e){}
     }
 }

class consumer extends Thread
{
    common c;
    consumer(common c)
    {     this.c=c;  }
    public void run()
     {
      try
         { c.consume();  }
         catch(Exception e){}
      }
 }
```

```
public class pc

{

  public static void main(String args[]) throws Exception

   {

        common c=new  common();
        producer p=new producer(c);
        consumer co=new consumer(c);
        p.start();

        co.start();

    }

}
```

9. Enumeration Program

```
public class DayOfWeek{

    public enum Day

  {

    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,

    THURSDAY, FRIDAY, SATURDAY

  }

    Day day;

    public DayOfWeek(Day day)

   {

    this.day = day;

    System.out.println(day);

    }
------------------------------------------------------------------------------------
----------------
     public boolean isWorkday()
   {
       if ((day == Day.MONDAY)||(Day.TUESDAY==day)||(day==Day.WEDNESDAY) ||
          (day==Day.THURSDAY) || (day==Day.FRIDAY))
       return true;
       else
       return false;
     }

     public static void main(String[] args)
    {
       DayOfWeek firstDay = new DayOfWeek(Day.FRIDAY);
       boolean a =firstDay.isWorkday();
       System.out.println("returned value is "+ a);
       DayOfWeek sixthDay = new DayOfWeek(Day.SUNDAY);
       boolean b=sixthDay.isWorkday();
       System.out.println("returned value is "+ b);
      }
    }
```

OUTPUT:

```
FRIDAY
returned value is true
SUNDAY
returned value is false
```

----

10. Write a JAVA Program which uses FileInputStream / FileOutPutStream Classes.

```
import java.io.*;
public class q8
{
      public static void main(String[] args)
      {
        try{
              int count=0;
              FileInputStream fis=new FileInputStream("/abc.txt");
               int avail=fis.available();
               byte[] b=new byte[avail];
               int done=fis.read(b);
            System.out.println("File Contents");
            for(byte a:b)
                        {
                                    Char  i=(char)a;
                                    System.out.print((char)a+"");
                         }
            FileOutputStream fos=new FileOutputstream("/xyz.txt");
            fos.write(b);
             }
            catch(Exception e){}
      }
 }
```

----

11. Write JAVA programs which demonstrates utilities of LinkedList Class

```
import java.util.*;

class Q7

 {

   public static void main(String args[])


   {


   LinkedList ll = new LinkedList();


     ll.add("F");
     ll.add("B");
     ll.add("D");
     ll.add("E");
     ll.add("C");
     ll.addLast("Z");
     ll.addFirst("A");
     ll.add(1, "A2");

     System.out.println("Original contents of linked list: " + ll);

     System.out.println("Index of first element " +  ll.indexOf("A"));


     ll.remove("F");

     ll.remove(2);

     System.out.println("Contents of linked list after deletion: " + ll);


     ll.removeFirst();

     ll.removeLast();

     System.out.println("linked list after deleting first and last: " + ll);


   // get and set a value

   Object val = ll.get(2);
     ll.set(2, "omega");

     System.out.println("linked List after change: " + ll);

   }

}
```