Practicals are not complete as prescribed by university.

All practicals are not in exact order as prescribed by mumbai university syllabus but they do implement all concepts suggested by university.

Every college has different way to conduct practicals so it is advised to consult with your college.

Kindly rectify errors if any.

Thanks

1A

```
// Design a java program for type casting different types of variables.
(implicit)


public class Test
{
    public static void main(String[] args)
    {
      int i = 100;
      long l = i;//no explicit type casting required
      float f = l;     //no explicit type casting required
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
    }

}
```

```
// Design a java program for type casting different types of variables.
(explicit)

public class Test
{
    public static void main(String[] args)
    {
      double d = 100.04;
      long l = (long)d;  //explicit type casting required
      int i = (int)l;   //explicit type casting required

      System.out.println("Double value "+d);
      System.out.println("Long value "+l);
      System.out.println("Int value "+i);

    }

}
```

1B

```
/* Design a Calculator class in java, and implement all the methods required
by calculator operations. */


//import Scanner as we require it.
import java.util.Scanner;

// the name of our class its public
public class SimpleCalculator {
    //void main
        public static void main (String[] args)
        {
            //declare int and char
            int a,b,result=0;
            char c;

            //Declare input as scanner
            Scanner input = new Scanner(System.in);

            //Take inputs
             System.out.println("Enter no. :");
             a = input.nextInt();
             System.out.println("Enter no. :");
             b = input.nextInt();
             System.out.println("Enter Operator :");
             String st = input.next();
             c = st.charAt(0);

             //add a switch statement
             switch(c)
             {
             case '+':
                 result = a+b;
                 System.out.println("Result = "+result);
                 break;
             case '-':
                 result = a-b;
                 System.out.println("Result = "+result);
                 break;
             case 'x':
                 result = a*b;
                 System.out.println("Result = "+result);
                 break;
             case '/':
                 result = a/b;
                 System.out.println("Result = "+result);
                 break;
             default:
                System.out.println("Syntax Error");
             }
        }
}
```

1C

```
/*
Design a java class for method overloading.
*/


class OverloadDemo
{
         void triangleArea(float base, float height)
         {
               float area;
               area = base * height / 2.0f;
               System.out.println(“Area = “ + Area);
         }

         void triangleArea(float side1, float side2, float side3)
         {
                  float area,s;
                  s = (side1 + side2 + side3) / 2.0;
                  area = Math.sqrt(s*(s-side1) * (s-side2) * (s-side3) );
                  System.out.println(“Area = “ + area);
         }
}

class MainOverloadDemo
{
            public static void main(String args[])
            {
                    OverloadDemo ovrldDemo = new OverloadDemo();
                    ovrldDemo.triangleArea(20.12,58.36);
                    ovrldDemo triangleArea(63.12,54.26,95.24);
            }
}
```

1C

```
/*
Design a java class for method overriding.
*/
// Source : http://www.careerride.com/java-method-overloading-and-
overriding.aspx



class Super
{
     int sum;
     A(int num1, int num2)
     {
             sum = a+b;
     }
     void add()
     {
            System.out.println("Sum : " + sum);
     }
}
class Sub extends Sub
{
       int subSum;
       Sub(int num1, int num2, int num3)
       {
               super(num1, num2);
               subSum = num1+num2+num3;
       }
       void add()
       {
               super.add();
               System.out.println("Sum of 3 nos : " +subSum);
       }
```

2A

```
/* Single Inheritance example program in Java */

Class A
{
   public void methodA()
   {
     System.out.println("Base class method");
   }
}

Class B extends A
{
   public void methodB()
   {
     System.out.println("Child class method");
   }
   public static void main(String args[])
   {
     B obj = new B();
     obj.methodA(); //calling super class method
     obj.methodB(); //calling local method
  }
}



/* Multilevel Inheritance example program in Java */

Class X
{
   public void methodX()
   {
     System.out.println("Class X method");
   }
}
Class Y extends X
{
public void methodY()
{
System.out.println("class Y method");
}
}
Class Z extends Y
{
   public void methodZ()
   {
     System.out.println("class Z method");
   }
   public static void main(String args[])
   {
     Z obj = new Z();
     obj.methodX(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method
  }
}
```

```
/* Hierarchical Inheritance example program in Java */

Class A
{
  public void methodA()
  {
     System.out.println("method of Class A");
  }
}
Class B extends A
{
  public void methodB()
  {
     System.out.println("method of Class B");
  }
}
Class C extends A
{
 public void methodC()
 {
 System.out.println("method of Class C");
 }
}
Class D extends A
{
  public void methodD()
  {
     System.out.println("method of Class D");
  }
}
Class MyClass
{
  public void methodB()
  {
     System.out.println("method of Class B");
  }
  public static void main(String args[])
  {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
     obj1.methodA();
     obj2.methodA();
     obj3.methodA();
  }
}
```

```
/* Hybrid Inheritance example program in Java */

public class A
{
     public void methodA()
     {
           System.out.println("Class A methodA");
      }
}
public class B extends A
{
      public void methodA()
      {
           System.out.println("Child class B is overriding inherited method
A");
      }
      public void methodB()
      {
           System.out.println("Class B methodB");
       }
}
public class C extends A
{
      public void methodA()
      {
           System.out.println("Child class C is overriding the methodA");
       }
      public void methodC()
      {
           System.out.println("Class C methodC");
       }
}
public class D extends B, C
{
      public void methodD()
      {
           System.out.println("Class D methodD");
       }
       public static void main(String args[])
       {
              D obj1= new D();
              obj1.methodD();
              obj1.methodA();
       }
}
```

2B

```
/* Design a java class for the use of interface. */

public class Main {

    public static void main(String[] args) {

          shape circleshape=new circle();

             circleshape.Draw();
    }
}

interface shape
 {
     public   String baseclass="shape";

     public void Draw();

 }

 class circle implements shape
 {

    public void Draw() {
        System.out.println("Drawing Circle here");
    }


 }
```

2C

```
public class AllStringFuctionExample {
  public static void main(String[] args) {
    String str = “All String function Example in java”;

    // convert string into Lower case
    String Lowercase = str.toLowerCase();
    System.out.println(“Lower case String ==> ” + Lowercase);

    // convert string into upper case
    String Uppercase = str.toUpperCase();
    System.out.println(“Upper case String ==> ” + Uppercase);

    // Find length of the given string
    System.out.println(“Length of the given string ==>” + str.length());

    // Trim the given string i.e. remove all first and last the spaces from
    // the string
    String tempstr = “    String trimming example   “;
    System.out.println(“String before trimming ==> ” + tempstr);
    System.out.println(“String after trimming ==> ” + tempstr.trim());

    // Find the character at the given index from the given string
    System.out.println(“Character at the index 6 is ==> ” + str.charAt(6));

    // find the substring between two index range
    System.out.println(“String between index 3 to 9 is ==> ”
    + str.substring(3, 9));

    // replace the character with another character
    System.out.println(“String after replacement ==> ”
    + str.replace(‘a’, ‘Y’));

    // replace the substring with another substring
    System.out.println(“String after replacement ==> ”
    + str.replace(“java”, “loan”));
  }
}
```

3A

```
/* Design a class in java to add two complex numbers using constructors. */


class Complex

{

                int iReal,iImaginary;
                Complex() //empty constructor

                {}

                Complex(int iTempReal,int iTempImaginary) // Two argument
constructor

                {
                                iReal=iTempReal;

                                iImaginary=iTempImaginary;
                }



                Complex fnAddComplex(Complex C1,Complex C2) // function to
add the complex numbers

                {
                                Complex CTemp=new Complex();
                                 CTemp.iReal=C1.iReal+C2.iReal;

                               CTemp.iImaginary=C1.iImaginary+C2.iImaginary;
                                return CTemp;
                }
}

class Complexmain
{
                public static void main(String[] a)
                {
                                Complex C1=new Complex(4,8); //calls the two
argument constructor

                                Complex C2=new Complex(5,7); //calls the two
argument constructor

                                Complex C3=new Complex();//calls the empty
constructor

                                C3=C3.fnAddComplex(C1,C2); //function call


                                //Display the results

                                System.out.println("---Sum---");

                                System.out.println("Real :" + C3.iReal);

                                System.out.println("Imaginary :" +
C3.iImaginary);

                }

}
```

3B

```
/*
Design a java class for performing all the matrix operations i.e addition,
multiplication,
transpose (etc >> =D).
*/

import java.util.Scanner;
interface Matrix
{
final static int M = 2, N = 2;
void readMatrix(); //Read a matrix
void displayMatrix(); //Display a matrix
void addMatrix(); //Add two matrices
void multMatrix(); // Multiply two matrices
void transposeMatrix(); //Transpose of matrix
}

class matrix1 implements Matrix
{
private int [ ][ ] a, b, c;
private int [ ][ ] read()
{
Scanner scan = new Scanner(System.in);
int [ ][ ] x = new int[M][N];
System.out.print("Enter elements of "+M+" x "+N+" matrix row-wise: ");
for(int i = 0; i < M; i++)
for(int j = 0; j < N; j++)
x[i][j] = scan.nextInt();
return x;
}
public void readMatrix()
{
a = read();
b = read();
}
private void display(int[ ][ ]x)
{
for(int i = 0; i < M; i++)
{
for(int j = 0; j < N; j++)
System.out.print(x[i][j]+" ");
System.out.println();
}
System.out.println();
}
public void displayMatrix()
{
display(a);
display(b);
display(c);
}
public void addMatrix()
{
c = new int[M][N];
for(int i = 0; i < M; i++)
for(int j = 0; j < N; j++)
c[i][j] = a[i][j] + b[i][j];
}
public void multMatrix()
{
c = new int[M][N];
for(int i = 0; i < M; i++)
for(int j = 0; j < N; j++)
for(int k = 0; k < M; k++)
c[i][j] += a[i][k] * b[k][j];
}
public void transposeMatrix()
{
c = new int[M][N];
for(int i = 0; i < M; i++)
for(int j = 0; j < N; j++)
c[j][i] = a[i][j];
}
}
public class Main153
{
public static void main(String[] args)
{
matrix1 z = new matrix1();
z.readMatrix();
z.addMatrix();
System.out.println("Addition");
z.displayMatrix();
z.multMatrix();
System.out.println("Multiplication");
z.displayMatrix();
z.transposeMatrix();
System.out.println("Transpose");
z.displayMatrix();
}
}
```

3C

```
/* Design a java class performing string operations. */

public class AllStringFuctionExample {
  public static void main(String[] args) {
    String str = “All String function Example in java”;

    // convert string into Lower case
    String Lowercase = str.toLowerCase();
    System.out.println(“Lower case String ==> ” + Lowercase);

    // convert string into upper case
    String Uppercase = str.toUpperCase();
    System.out.println(“Upper case String ==> ” + Uppercase);

    // Find length of the given string
    System.out.println(“Length of the given string ==>” + str.length());

    // Trim the given string i.e. remove all first and last the spaces
from
    // the string
    String tempstr = “    String trimming example   “;
    System.out.println(“String before trimming ==> ” + tempstr);
    System.out.println(“String after trimming ==> ” + tempstr.trim());

    // Find the character at the given index from the given string
    System.out.println(“Character at the index 6 is ==> ” +
str.charAt(6));

    // find the substring between two index range
    System.out.println(“String between index 3 to 9 is ==> ”
    + str.substring(3, 9));

    // replace the character with another character
    System.out.println(“String after replacement ==> ”
    + str.replace(‘a’, ‘Y’));

    // replace the substring with another substring
    System.out.println(“String after replacement ==> ”
    + str.replace(“java”, “loan”));
  }
}
```

4A

```
/* Design a java class for implementing the concept of threading and
multithreading. */

class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}

public class TestThread {
   public static void main(String args[]) {

      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

4B

```
/* Design a java class for performing all the file-operations. */

import java.io.*;

public class CopyFile {
   public static void main(String args[]) throws IOException
   {
      FileInputStream in = null;
      FileOutputStream out = null;

      try {
         in = new FileInputStream("input.txt");
         out = new FileOutputStream("output.txt");

         int c;
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      }finally {
         if (in != null) {
            in.close();
         }
         if (out != null) {
            out.close();
         }
      }
   }
}
```

4C

```
/* Design a java class for operating the random access files (using =D) */


import java.io.*;

class  RandRW
{
    publicstaticvoid main(String[] args)
    {
         RandomAccessFile file = null;
         try{
             file = new RandomAccessFile("rand.txt","rw");

             //Writing to the file
             file.writeChar('V');
             file.writeInt(999);
             file.writeDouble(99.99);

             file.seek(0);     //Go to the begining//Reading from the file
             System.out.println(file.readChar());
             System.out.println(file.readInt());
             System.out.println(file.readDouble());

             file.seek(2);  //Go to the Second Item
             System.out.println(file.readInt());

             //Go to the end and append false to the file
             file.seek(file.length());
             file.writeBoolean(true);

             file.seek(4);
             System.out.println(file.readBoolean());
             file.close();
            }catch(Exception e) {}
        }}
```

5A

```
/* Design a class for sorting the names or numbers in ascending and
descending order. */

import java.util.Arrays;
import java.util.Collections;

public class HashtableDemo {

public static void main(String args[]) {
String[] companies = { "Google", "Apple", "Sony" };

// sorting java array in ascending order
System.out.println("Sorting String Array in Ascending order in Java
Example");
System.out.println("Unsorted String Array in Java: ");
printNumbers(companies);
Arrays.sort(companies);
System.out.println("Sorted String Array in ascending order : ");
printNumbers(companies);

// sorting java array in descending order
System.out.println("Sorting Array in Descending order in Java Example");
System.out.println("Unsorted int Array in Java: ");
printNumbers(companies);
Arrays.sort(companies, Collections.reverseOrder());
System.out.println("Sorted int Array in descending order : ");
printNumbers(companies);

System.out.println("Sorting part of array in java:");
int[] numbers = { 1, 3, 2, 5, 4 };
Arrays.sort(numbers, 0, 3);
System.out.println("Sorted sub array in Java: ");
for (int num : numbers) {
System.out.println(num);
}

}

public static void printNumbers(String[] companies) {
for (String company : companies) {
System.out.println(company);
}
}

}
```

5B

```
/* Design a java class for implementing the operations of stack. */

import java.util.*;
class StackDemo {
static void showpush(Stack st, int a) {
st.push(new Integer(a));
System.out.println("push(" + a + ")");
System.out.println("stack: " + st);
}
static void showpop(Stack st) {
System.out.print("pop -> ");
Integer a = (Integer) st.pop();
System.out.println(a);
System.out.println("stack: " + st);
}
public static void main(String args[]) {
Stack st = new Stack();
System.out.println("stack: " + st);
showpush(st, 42);
showpush(st, 66);
showpush(st, 99);
showpop(st);
showpop(st);
showpop(st);
try {
showpop(st);
} catch (EmptyStackException e) {
System.out.println("empty stack");
}
}
}
```

6A

```
/* Design a java class for implementing the operations of stack. */

import java.util.*;
class StackDemo {
static void showpush(Stack st, int a) {
st.push(new Integer(a));
System.out.println("push(" + a + ")");
System.out.println("stack: " + st);
}
static void showpop(Stack st) {
System.out.print("pop -> ");
Integer a = (Integer) st.pop();
System.out.println(a);
System.out.println("stack: " + st);
}
public static void main(String args[]) {
Stack st = new Stack();
System.out.println("stack: " + st);
showpush(st, 42);
showpush(st, 66);
showpush(st, 99);
showpop(st);
showpop(st);
showpop(st);
try {
showpop(st);
} catch (EmptyStackException e) {
System.out.println("empty stack");
}
}
}
```

6B

```
/* Design a class in java for implementing the operations of circular queue.
*/



import java.io.*;
class circularQ
{
int Q[] = new int[100];
int n, front, rear;
static BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
public circularQ(int nn)
{
n=nn;
front = rear = 0;
}
public void add(int v)
{
if((rear+1) % n != front)
{
rear = (rear+1)%n;
Q[rear] = v;
}
else
System.out.println("Queue is full !");
}
public int del()
{
int v;
if(front!=rear)
{
front = (front+1)%n;
v = Q[front];
return v;
}
else
return -9999;
}
public void disp()
{
int i;
if(front != rear)
{
i = (front +1) %n;
while(i!=rear)
{
System.out.println(Q[i]);
i = (i+1) % n;
}
}
else
System.out.println("Queue is empty !");
}
public static void main() throws IOException
{
System.out.print("Enter the size of the queue : ");
int size = Integer.parseInt(br.readLine());
circularQ call = new circularQ(size);
int choice;
boolean exit = false;
while(!exit)
{
System.out.print("\n1 : Add\n2 : Delete\n3 : Display\n4 :
Exit\n\nYour Choice : ");
choice = Integer.parseInt(br.readLine());
switch(choice)
{
case 1 :
System.out.print("\nEnter number to be added : ");
int num = Integer.parseInt(br.readLine());
call.add(num);
break;
case 2 :
int popped = call.del();
if(popped != -9999)
System.out.println("\nDeleted : " +popped);
else
System.out.println("\nQueue is empty !");
break;
case 3 :
call.disp();
break;
case 4 :
exit = true;
break;
default :
System.out.println("\nWrong Choice !");
break;
}
}
}
}
```

7A

```
/* Design a class to implement the operations of singly link-list. (
insertion , deletion, sorting, display) */


import java.util.Scanner;


/*  Class Node  */

class Node

{

    protected int data;

    protected Node link;


    /*  Constructor  */

    public Node()

    {

        link = null;

        data = 0;

    }

    /*  Constructor  */

    public Node(int d,Node n)

    {

        data = d;

        link = n;

    }

    /*  Function to set link to next Node  */

    public void setLink(Node n)

    {

        link = n;

    }

    /*  Function to set data to current Node  */

    public void setData(int d)

    {

        data = d;

    }

    /*  Function to get link to next node  */
    public Node getLink()
    {
        return link;
    }
    /*  Function to get data from current Node  */
    public int getData()
    {
        return data;
    }
}

/* Class linkedList */
class linkedList
{
    protected Node start;
    protected Node end ;
    public int size ;

    /*  Constructor  */
    public linkedList()
    {
        start = null;
        end = null;
        size = 0;
    }
    /*  Function to check if list is empty  */
    public boolean isEmpty()
    {
        return start == null;
    }
    /*  Function to get size of list  */
    public int getSize()
    {
        return size;
    }
```

```
}

/*  Function to insert an element at begining  */
public void insertAtStart(int val)
{
    Node nptr = new Node(val, null);
    size++ ;
    if(start == null)
    {
        start = nptr;
        end = start;
    }
    else
    {
        nptr.setLink(start);
        start = nptr;
    }
}
/*  Function to insert an element at end  */
public void insertAtEnd(int val)
{
    Node nptr = new Node(val,null);
    size++ ;
    if(start == null)
    {
        start = nptr;
        end = start;
    }
    else
    {
        end.setLink(nptr);
        end = nptr;
    }
}
/*  Function to insert an element at position  */
public void insertAtPos(int val , int pos)
```

```
{
    Node nptr = new Node(val, null);
    Node ptr = start;
    pos = pos - 1 ;
    for (int i = 1; i < size; i++)
    {
        if (i == pos)
        {
            Node tmp = ptr.getLink() ;
            ptr.setLink(nptr);
            nptr.setLink(tmp);
            break;
        }
        ptr = ptr.getLink();
    }
    size++ ;
}
/*  Function to delete an element at position  */
public void deleteAtPos(int pos)
{
    if (pos == 1)
    {
        start = start.getLink();
        size--;
        return ;
    }
    if (pos == size)
    {
        Node s = start;
        Node t = start;
        while (s != end)
        {
            t = s;
            s = s.getLink();
        }
```

```
            end = t;
            end.setLink(null);
            size --;
            return;
        }
        Node ptr = start;
        pos = pos - 1 ;
        for (int i = 1; i < size - 1; i++)
        {
            if (i == pos)
            {
                Node tmp = ptr.getLink();
                tmp = tmp.getLink();
                ptr.setLink(tmp);
                break;
            }
            ptr = ptr.getLink();
        }
        size-- ;
    }
    /*  Function to display elements  */
    public void display()
    {
        System.out.print("\nSingly Linked List = ");
        if (size == 0)
        {
            System.out.print("empty\n");
            return;
        }
        if (start.getLink() == null)
        {
            System.out.println(start.getData() );
            return;
        }
        Node ptr = start;
        System.out.print(start.getData()+ "->");
        ptr = start.getLink();
        while (ptr.getLink() != null)
        {
            System.out.print(ptr.getData()+ "->");
            ptr = ptr.getLink();
        }
        System.out.print(ptr.getData()+ "\n");
    }
}

/*  Class SinglyLinkedList  */
public class SinglyLinkedList
{
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        /* Creating object of class linkedList */
        linkedList list = new linkedList();
        System.out.println("Singly Linked List Test\n");
        char ch;
        /*  Perform list operations  */
        do
        {
            System.out.println("\nSingly Linked List Operations\n");
            System.out.println("1. insert at begining");
            System.out.println("2. insert at end");
            System.out.println("3. insert at position");
            System.out.println("4. delete at position");
            System.out.println("5. check empty");
            System.out.println("6. get size");
            int choice = scan.nextInt();
            switch (choice)
            {
```

```
case 1 :
    System.out.println("Enter integer element to insert");
    list.insertAtStart( scan.nextInt() );
    break;
case 2 :
    System.out.println("Enter integer element to insert");
    list.insertAtEnd( scan.nextInt() );
    break;
case 3 :
    System.out.println("Enter integer element to insert");
    int num = scan.nextInt() ;
    System.out.println("Enter position");
    int pos = scan.nextInt() ;
    if (pos <= 1 || pos > list.getSize() )
        System.out.println("Invalid position\n");
    else
        list.insertAtPos(num, pos);
    break;
case 4 :
    System.out.println("Enter position");
    int p = scan.nextInt() ;
    if (p < 1 || p > list.getSize() )
        System.out.println("Invalid position\n");
    else
        list.deleteAtPos(p);
    break;
case 5 :
    System.out.println("Empty status = "+ list.isEmpty());
    break;
case 6 :
    System.out.println("Size = "+ list.getSize() +" \n");
    break;
 default :
    System.out.println("Wrong Entry \n ");
```

```
                break;
            }
            /*  Display List  */
            list.display();
            System.out.println("\nDo you want to continue (Type y or n)
\n");
            ch = scan.next().charAt(0);
        } while (ch == 'Y'|| ch == 'y');
    }
}
```

7B

```
/* Design a class to implement the operations of doubly-linked list. */

import java.util.Scanner;


/*  Class Node  */
class Node
{
    protected int data;
    protected Node next, prev;

    /* Constructor */
    public Node()
    {
        next = null;
        prev = null;
        data = 0;
    }
    /* Constructor */
    public Node(int d, Node n, Node p)
    {
        data = d;
        next = n;
        prev = p;
    }
    /* Function to set link to next node */
    public void setLinkNext(Node n)
    {
        next = n;
    }
    /* Function to set link to previous node */
    public void setLinkPrev(Node p)
    {
        prev = p;
    }
    /* Funtion to get link to next node */
    public Node getLinkNext()
    {
        return next;
    }
    /* Function to get link to previous node */
    public Node getLinkPrev()
    {
        return prev;
    }
    /* Function to set data to node */
    public void setData(int d)
    {
        data = d;
    }
    /* Function to get data from node */
    public int getData()
    {
        return data;
    }
}

/* Class linkedList */
class linkedList
{
    protected Node start;
    protected Node end ;
    public int size;

    /* Constructor */
    public linkedList()
    {
        start = null;
        end = null;
        size = 0;
```

```
}
/* Function to check if list is empty */
public boolean isEmpty()
{
    return start == null;
}
/* Function to get size of list */
public int getSize()
{
    return size;
}
/* Function to insert element at begining */
public void insertAtStart(int val)
{
    Node nptr = new Node(val, null, null);
    if(start == null)
    {
        start = nptr;
        end = start;
    }
    else
    {
        start.setLinkPrev(nptr);
        nptr.setLinkNext(start);
        start = nptr;
    }
    size++;
}
/* Function to insert element at end */
public void insertAtEnd(int val)
{
    Node nptr = new Node(val, null, null);
    if(start == null)
    {
        start = nptr;
        end = start;
    }
    else
    {
        nptr.setLinkPrev(end);
        end.setLinkNext(nptr);
        end = nptr;
    }
    size++;
}
/* Function to insert element at position */
public void insertAtPos(int val , int pos)
{
    Node nptr = new Node(val, null, null);
    if (pos == 1)
    {
        insertAtStart(val);
        return;
    }
    Node ptr = start;
    for (int i = 2; i <= size; i++)
    {
        if (i == pos)
        {
            Node tmp = ptr.getLinkNext();
            ptr.setLinkNext(nptr);
            nptr.setLinkPrev(ptr);
            nptr.setLinkNext(tmp);
            tmp.setLinkPrev(nptr);
        }
        ptr = ptr.getLinkNext();
    }
    size++ ;
}
```

```
/* Function to delete node at position */
public void deleteAtPos(int pos)
{
    if (pos == 1)
    {
        if (size == 1)
        {
            start = null;
            end = null;
            size = 0;
            return;
        }
        start = start.getLinkNext();
        start.setLinkPrev(null);
        size--;
        return ;
    }
    if (pos == size)
    {
        end = end.getLinkPrev();
        end.setLinkNext(null);
        size-- ;
    }
    Node ptr = start.getLinkNext();
    for (int i = 2; i <= size; i++)
    {
        if (i == pos)
        {
            Node p = ptr.getLinkPrev();
            Node n = ptr.getLinkNext();

            p.setLinkNext(n);
            n.setLinkPrev(p);
            size-- ;
```

```
                return;
            }
            ptr = ptr.getLinkNext();
        }
    }
    /* Function to display status of list */
    public void display()
    {
        System.out.print("\nDoubly Linked List = ");
        if (size == 0)
        {
            System.out.print("empty\n");
            return;
        }
        if (start.getLinkNext() == null)
        {
            System.out.println(start.getData() );
            return;
        }
        Node ptr = start;
        System.out.print(start.getData()+ " <-> ");
        ptr = start.getLinkNext();
        while (ptr.getLinkNext() != null)
        {
            System.out.print(ptr.getData()+ " <-> ");
            ptr = ptr.getLinkNext();
        }
        System.out.print(ptr.getData()+ "\n");
    }
}

/* Class DoublyLinkedList */
public class DoublyLinkedList
{
    public static void main(String[] args)
```

```
{
    Scanner scan = new Scanner(System.in);
    /* Creating object of linkedList */
    linkedList list = new linkedList();
    System.out.println("Doubly Linked List Test\n");
    char ch;
    /*  Perform list operations  */
    do
    {
        System.out.println("\nDoubly Linked List Operations\n");
        System.out.println("1. insert at begining");
        System.out.println("2. insert at end");
        System.out.println("3. insert at position");
        System.out.println("4. delete at position");
        System.out.println("5. check empty");
        System.out.println("6. get size");

        int choice = scan.nextInt();
        switch (choice)
        {
        case 1 :
            System.out.println("Enter integer element to insert");
            list.insertAtStart( scan.nextInt() );
            break;
        case 2 :
            System.out.println("Enter integer element to insert");
            list.insertAtEnd( scan.nextInt() );
            break;
        case 3 :
            System.out.println("Enter integer element to insert");
            int num = scan.nextInt() ;
            System.out.println("Enter position");
            int pos = scan.nextInt() ;
            if (pos < 1 || pos > list.getSize() )
```

```
                    System.out.println("Invalid position\n");
                else
                    list.insertAtPos(num, pos);
                break;
            case 4 :
                System.out.println("Enter position");
                int p = scan.nextInt() ;
                if (p < 1 || p > list.getSize() )
                    System.out.println("Invalid position\n");
                else
                    list.deleteAtPos(p);
                break;
            case 5 :
                System.out.println("Empty status = "+ list.isEmpty());
                break;
            case 6 :
                System.out.println("Size = "+ list.getSize() +" \n");
                break;
            default :
                System.out.println("Wrong Entry \n ");
                break;
            }
            /*  Display List  */
            list.display();
            System.out.println("\nDo you want to continue (Type y or n) \n");
            ch = scan.next().charAt(0);

        } while (ch == 'Y'|| ch == 'y');
    }
}
```

8A

```java
/* Implement the concept of hashing technique. */

import java.util.*;

public class HashTableDemo {

   public static void main(String args[]) {
      // Create a hash map
      Hashtable balance = new Hashtable();
      Enumeration names;
      String str;
      double bal;

      balance.put("Zara", new Double(3434.34));
      balance.put("Mahnaz", new Double(123.22));
      balance.put("Ayan", new Double(1378.00));
      balance.put("Daisy", new Double(99.22));
      balance.put("Qadir", new Double(-19.08));

      // Show all balances in hash table.
      names = balance.keys();
      while(names.hasMoreElements()) {
         str = (String) names.nextElement();
         System.out.println(str + ": " +
         balance.get(str));
      }
      System.out.println();
      // Deposit 1,000 into Zara's account
      bal = ((Double)balance.get("Zara")).doubleValue();
      balance.put("Zara", new Double(bal+1000));
      System.out.println("Zara's new balance: " +
      balance.get("Zara"));
   }
}
```

8B

```
/* Design a class to create a tree and also implement the binary search
tree. */


class BinaryTree{
    public static void main(String[] a){
      System.out.println(new BT().Start());
    }
}


// This class invokes the methods to create a tree,
// insert, delete and serach for  elements on it
class BT {

    public int Start(){
      Tree root ;
      boolean ntb ;
      int nti ;

      root = new Tree();
      ntb = root.Init(16);
      ntb = root.Print();
      System.out.println(100000000);
      ntb = root.Insert(8) ;
      ntb = root.Print();
      ntb = root.Insert(24) ;
      ntb = root.Insert(4) ;
      ntb = root.Insert(12) ;
      ntb = root.Insert(20) ;
      ntb = root.Insert(28) ;
      ntb = root.Insert(14) ;
      ntb = root.Print();
      System.out.println(root.Search(24));
      System.out.println(root.Search(12));
      System.out.println(root.Search(16));
      System.out.println(root.Search(50));
      System.out.println(root.Search(12));
      ntb = root.Delete(12);
      ntb = root.Print();
      System.out.println(root.Search(12));

      return 0 ;
    }

}

class Tree{
    Tree left ;
    Tree right;
    int key ;
    boolean has_left ;
    boolean has_right ;
    Tree my_null ;

    // Initialize a node with a key value and no children
    public boolean Init(int v_key){
      key = v_key ;
      has_left = false ;
      has_right = false ;
      return true ;
    }

    // Update the right child with rn
    public boolean SetRight(Tree rn){
      right = rn ;
      return true ;
```

```
}

// Update the left child with ln
public boolean SetLeft(Tree ln){
  left = ln ;
  return true ;
}

public Tree GetRight(){
  return right ;
}

public Tree GetLeft(){
  return left;
}

public int GetKey(){
  return key ;
}

public boolean SetKey(int v_key){
  key = v_key ;
  return true ;
}

public boolean GetHas_Right(){
  return has_right ;
}

public boolean GetHas_Left(){
  return has_left ;
}

public boolean SetHas_Left(boolean val){
   has_left = val ;
   return true ;
}

public boolean SetHas_Right(boolean val){
   has_right = val ;
   return true ;
}

// This method compares two integers and
// returns true if they are equal and false
// otherwise
public boolean Compare(int num1 , int num2){
  boolean ntb ;
  int nti ;

  ntb = false ;
  nti = num2 + 1 ;
  if (num1 < num2) ntb = false ;
  else if (!(num1 < nti)) ntb = false ;
  else ntb = true ;
  return ntb ;
}


// Insert a new element in the tree
public boolean Insert(int v_key){
  Tree new_node ;
  boolean ntb ;
  boolean cont ;
  int key_aux ;
  Tree current_node ;

  new_node = new Tree();
  ntb = new_node.Init(v_key) ;
```

```
    current_node = this ;
    cont = true ;
    while (cont){
        key_aux = current_node.GetKey();
        if (v_key < key_aux){
          if (current_node.GetHas_Left())
              current_node = current_node.GetLeft() ;
          else {
              cont = false ;
              ntb = current_node.SetHas_Left(true);
              ntb = current_node.SetLeft(new_node);
          }
        }
        else{
          if (current_node.GetHas_Right())
              current_node = current_node.GetRight() ;
          else {
              cont = false ;
              ntb = current_node.SetHas_Right(true);
              ntb = current_node.SetRight(new_node);
          }
        }
    }
    return true ;
  }


  // Delete an element from the tree
  public boolean Delete(int v_key){
    Tree current_node ;
    Tree parent_node ;
    boolean cont ;
    boolean found ;
    boolean is_root ;
    int key_aux ;
    boolean ntb ;

    current_node = this ;
    parent_node = this ;
    cont = true ;
    found = false ;
    is_root = true ;
    while (cont){
        key_aux = current_node.GetKey();
        if (v_key < key_aux)
          if (current_node.GetHas_Left()){
              parent_node = current_node ;
              current_node = current_node.GetLeft() ;
          }
          else cont = false ;
        else
          if (key_aux < v_key)
              if (current_node.GetHas_Right()){
                parent_node = current_node ;
                current_node = current_node.GetRight() ;
              }
              else cont = false ;
          else {
              if (is_root)
                if ((!current_node.GetHas_Right()) &&
                    (!current_node.GetHas_Left()) )
                    ntb = true ;
                else
                    ntb = this.Remove(parent_node,current_node);
              else ntb = this.Remove(parent_node,current_node);
              found = true ;
              cont = false ;
          }
        is_root = false ;
    }
    return found ;
  }


  // Check if the element to be removed will use the
  // righ or left subtree if one exists
  public boolean Remove(Tree p_node, Tree c_node){
    boolean ntb ;
    int auxkey1 ;
    int auxkey2 ;

    if (c_node.GetHas_Left())
        ntb = this.RemoveLeft(p_node,c_node) ;
    else
        if (c_node.GetHas_Right())
          ntb = this.RemoveRight(p_node,c_node) ;
        else {
          auxkey1 = c_node.GetKey();
          //auxtree01 = p_node.GetLeft() ;
          //auxkey2 = auxtree01.GetKey() ;
          auxkey2 = (p_node.GetLeft()).GetKey() ;
          if (this.Compare(auxkey1,auxkey2)) {
              ntb = p_node.SetLeft(my_null);
              ntb = p_node.SetHas_Left(false);
          }
          else {
              ntb = p_node.SetRight(my_null);
              ntb = p_node.SetHas_Right(false);
          }
        }
    return true ;
  }


  // Copy the child key to the parent until a leaf is
  // found and remove the leaf. This is done with the
  // right subtree
  public boolean RemoveRight(Tree p_node, Tree c_node){
    boolean ntb ;

    while (c_node.GetHas_Right()){
        //auxtree01 = c_node.GetRight() ;
        //auxint02 = auxtree01.GetKey();
        //ntb = c_node.SetKey(auxint02);
        ntb = c_node.SetKey((c_node.GetRight()).GetKey());
        p_node = c_node ;
        c_node = c_node.GetRight() ;
    }
    ntb = p_node.SetRight(my_null);
    ntb = p_node.SetHas_Right(false);
    return true ;
  }


  // Copy the child key to the parent until a leaf is
  // found and remove the leaf. This is done with the
  // left subtree
  public boolean RemoveLeft(Tree p_node, Tree c_node){
    boolean ntb ;

    while (c_node.GetHas_Left()){
        //auxtree01 = c_node.GetLeft() ;
        //auxint02 = auxtree01.GetKey();
        //ntb = c_node.SetKey(auxint02);
        ntb = c_node.SetKey((c_node.GetLeft()).GetKey());
        p_node = c_node ;
        c_node = c_node.GetLeft() ;
    }
```

```
      ntb = p_node.SetLeft(my_null);
      ntb = p_node.SetHas_Left(false);
      return true ;
    }

    // Search for an elemnt in the tree
    public int Search(int v_key){
      boolean cont ;
      int ifound ;
      Tree current_node;
      int key_aux ;

      current_node = this ;
      cont = true ;
      ifound = 0 ;
      while (cont){
          key_aux = current_node.GetKey();
          if (v_key < key_aux)
            if (current_node.GetHas_Left())
                current_node = current_node.GetLeft() ;
            else cont = false ;
          else
            if (key_aux < v_key)
                if (current_node.GetHas_Right())
                  current_node = current_node.GetRight() ;
                else cont = false ;
            else {
                ifound = 1 ;
                cont = false ;
            }
      }
      return ifound ;
    }

    // Invoke the method to really print the tree elements
    public boolean Print(){
      Tree current_node;
      boolean ntb ;

      current_node = this ;
      ntb = this.RecPrint(current_node);
      return true ;
    }

    // Print the elements of the tree
    public boolean RecPrint(Tree node){
      boolean ntb ;

      if (node.GetHas_Left()){
          //auxtree01 = node.GetLeft() ;
          //ntb = this.RecPrint(auxtree01);
          ntb = this.RecPrint(node.GetLeft());
      } else ntb = true ;
      System.out.println(node.GetKey());
      if (node.GetHas_Right()){
          //auxtree01 = node.GetRight() ;
          //ntb = this.RecPrint(auxtree01);
          ntb = this.RecPrint(node.GetRight());
      } else ntb = true ;
      return true ;
    }

}
```

9A

```
/* Heap Sort */
```

```
public class HeapSort
{
    private static int[] a;
    private static int n;
    private static int left;
    private static int right;
    private static int largest;


    public static void buildheap(int []a){
        n=a.length-1;
        for(int i=n/2;i>=0;i--){
            maxheap(a,i);
        }
    }

    public static void maxheap(int[] a, int i){
        left=2*i;
        right=2*i+1;
        if(left <= n && a[left] > a[i]){
            largest=left;
        }
        else{
            largest=i;
        }

        if(right <= n && a[right] > a[largest]){
            largest=right;
        }
        if(largest!=i){
            exchange(i,largest);
            maxheap(a, largest);
        }
    }

    public static void exchange(int i, int j){
        int t=a[i];
        a[i]=a[j];
        a[j]=t;
        }

    public static void sort(int []a0){
        a=a0;
        buildheap(a);

        for(int i=n;i>0;i--){
            exchange(0, i);
            n=n-1;
            maxheap(a, 0);
        }
    }

    public static void main(String[] args) {
        int []a1={4,1,3,2,16,9,10,14,8,7};
        sort(a1);
        for(int i=0;i<a1.length;i++){
            System.out.print(a1[i] + " ");
        }
    }
}
```

9B1

```
/* Design a class in java for implementing insertion sort. */

 public class InsertionSort{
  public static void main(String a[]){
  int i;
  int array[] = {12,9,4,99,120,1,3,10};
  System.out.println("\n\n RoseIndia\n\n");
  System.out.println(" Selection Sort\n\n");
  System.out.println("Values Before the sort:\n");
  for(i = 0; i < array.length; i++)
  System.out.print( array[i]+"  ");
  System.out.println();
  insertion_srt(array, array.length);
  System.out.print("Values after the sort:\n");
  for(i = 0; i <array.length; i++)
  System.out.print(array[i]+"  ");
  System.out.println();
  System.out.println("PAUSE");
  }

  public static void insertion_srt(int array[], int n){
  for (int i = 1; i < n; i++){
  int j = i;
  int B = array[i];
  while ((j > 0) && (array[j-1] > B)){
  array[j] = array[j-1];
  j--;
  }
  array[j] = B;
  }
  }
}
```

9B2

```
/* Design a class in java for implementing selection sort */

public static void selectionSort1(int[] x)
 {
    for (int i=0; i<x.length-1; i++)
 {
        for (int j=i+1; j<x.length; j++)
 {
            if (x[i] > x[j])
 {
                //... Exchange elements
                int temp = x[i];
                x[i] = x[j];
                x[j] = temp;
            }
        }
    }
}
```

10A1

```
/* Design a class in java for bubble sort */


public class BubbleSort {

        public static void main(String[] args) {

                //create an int array we want to sort using bubble sort
algorithm
                int intArray[] = new int[]{5,90,35,45,150,3};

                //print array before sorting using bubble sort algorithm
                System.out.println("Array Before Bubble Sort");
                for(int i=0; i < intArray.length; i++){
                        System.out.print(intArray[i] + " ");
                }

                //sort an array using bubble sort algorithm
                bubbleSort(intArray);

                System.out.println("");

                //print array after sorting using bubble sort algorithm
                System.out.println("Array After Bubble Sort");
                for(int i=0; i < intArray.length; i++){
                        System.out.print(intArray[i] + " ");
                }

        }

        private static void bubbleSort(int[] intArray) {

                /*
                 * In bubble sort, we basically traverse the array from
first
                 * to array_length - 1 position and compare the element with
the next one.
                 * Element is swapped with the next element if the next
element is greater.
                 *
                 * Bubble sort steps are as follows.
                 *
                 * 1. Compare array[0] & array[1]
                 * 2. If array[0] > array [1] swap it.
                 * 3. Compare array[1] & array[2]
                 * 4. If array[1] > array[2] swap it.
                 * ...
                 * 5. Compare array[n-1] & array[n]
                 * 6. if [n-1] > array[n] then swap it.
                 *
                 * After this step we will have largest element at the last
index.
                 *
                 * Repeat the same steps for array[1] to array[n-1]
                 *
                 */

                int n = intArray.length;
                int temp = 0;

                for(int i=0; i < n; i++){
                        for(int j=1; j < (n-i); j++){

                                if(intArray[j-1] > intArray[j]){
                                        //swap the elements!
                                        temp = intArray[j-1];
                                        intArray[j-1] = intArray[j];
                                        intArray[j] = temp;
                                }

                        }
                }

        }
}
```

10A2

```
/* Design a class in java for implementing the graph */


import java.util.*;
import java.io.*;

public class GraphTest {
    public static void main( String [] args ) {
        test1();
        test2();
    }
    ////////////////////////
    private static void test1() {
        for ( int nodeCount = 0; nodeCount <  5; ++nodeCount ) {
            for ( int edgeCount = -1; edgeCount <= nodeCount * 10;
++edgeCount ) {
                try {
                    System.out.println( "\n----- Test case nodeCount: " +
nodeCount + " edgeCount: " + edgeCount + " -----" );
                    System.out.flush();
                    test( nodeCount, edgeCount, true );
                }
                catch ( Exception e ) {
                    System.out.println( "Graph creation failed: " +
e.getMessage() );
                }
            }
        }
        System.out.flush();
    }
    ////////////////////////
    private static void test2() {
        int nodeCount = 10000;
        int edgeCount = nodeCount * 10;
        System.out.println( "\n----- Test case nodeCount: " + nodeCount + "
edgeCount: " + edgeCount + " -----" );
        test( nodeCount, edgeCount, false );
    }
    ////////////////////////
    private static void test( int nodeCount, int edgeCount, boolean
dumpGraph ) {
        Graph rg = Graph.createRandomGraph( nodeCount, edgeCount );
        if ( !dumpGraph )
            System.out.print( rg.getGraphSummary() );
        // Dump degree historgram
        int maxDegree = rg.computeMaxDegree();
        for ( int i = 0; i  <= maxDegree; ++i ) {
            int nodeCountWithDegree = rg.countNodesWithDegree( i );
            System.out.println( "Nodes with degree " + i + ": " +
nodeCountWithDegree );
        }
        // Test for self-looping nodes
        System.out.println( "Exists self-loops: " + rg.hasSelfLoops() );
        if ( dumpGraph )
            System.out.print( rg.toStringVerbose() );
    }
}

/////////////////////////////////////////////////////////////////////////////
//

final class Graph {

    private SortedMap< String, Node > nodeMap = null;
    private Map< String, Edge > edgeMap = null;

    public Graph() {
        nodeMap = new TreeMap< String, Node >( new Comparator< String >() {
            public int compare( String s1, String s2 ) {
                return s1.compareTo( s2 );
            }
        } );
        edgeMap = new HashMap< String, Edge >();
    }
    ////////////////////////
    public static Graph createRandomGraph( int nodeCount, int edgeCount ) {
        if ( nodeCount < 1 || edgeCount < 0 ) throw new
IllegalArgumentException( "nodeCount must be >= 1 and edgeCount must be >=
0!" );
        Random rnGen = new Random( System.currentTimeMillis() );
        int maxEdges = getMaxEdgesForGraph( nodeCount );
        if ( edgeCount > maxEdges )
            throw new IllegalArgumentException( "Input edgeCount (" +
edgeCount + ") exceeds maximum possible edges for graph with " + nodeCount +
" nodes!" );
        // Create empty Graph object
        Graph g = new Graph();
        // Create temp array to hold node keys - required for
getRandomEdge()
        String [] nodeKeys = new String [ nodeCount ];
        // Create and add nodeList
        for ( int i = 0; i < nodeCount; ++i ) {
            String nodeId = Integer.toString( g.getNodeCount() );
            nodeKeys[ i ] = nodeId;
            Node n = new Node( nodeId );
            g.addNode( n ); // Let list index be node's id
        }
        // Create and add edgeList
        for ( int i = 0; i < edgeCount; ++i ) {
            Edge e = Graph.getRandomEdge( rnGen, g, nodeKeys );
            g.addEdge( e );
        }
        return g;
    }
    ////////////////////////
    public void addNode( Node n ) {
        if ( n == null ) throw new IllegalArgumentException( "Argument must
be non-null!" );
        if ( nodeMap.get( n.getId() ) != null ) throw new
IllegalArgumentException( "Attempt to add node with duplicate id <" +
n.getId() + ">" );
        nodeMap.put( n.getId(), n);
    }
    ////////////////////////
    private static Edge getRandomEdge( Random rnGen, Graph g, String [] keys
) {
        if ( g.getNodeCount() < 2 ) throw new IllegalStateException(
"Attempt to add edge when < 2 nodes are in graph!" );
        if ( keys == null || keys.length != g.getNodeCount() ) throw new
IllegalArgumentException( "keys argument null or wrong size!" );
        Node n1 = null;
        Node n2 = null;
        Edge retEdge = null;
        while ( true ) {
            n1 = g.nodeMap.get( keys[ rnGen.nextInt( g.getNodeCount() ) ] );
            n2 = g.nodeMap.get( keys[ rnGen.nextInt( g.getNodeCount() ) ] );
            if ( n1 == n2 ) // Skip if already have edge between these two
nodes
                continue;
            String id = Edge.computeDefaultEdgeId( n1, n2 );
            if ( g.edgeMap.get( id ) != null )
                continue;
            retEdge = new Edge( n1, n2, id );
            break;
        }
        return retEdge;
    }
    ////////////////////////
    public void addEdge( Edge e ) {
        if ( edgeMap.get( e.getId() ) != null ) throw new
IllegalStateException( "Attemp to add edge wiith duplicate id <" + e.getId()
+ ">" );
        edgeMap.put( e.getId(), e );
        e.getN1().incrementDegree();
        e.getN2().incrementDegree();
    }
    ////////////////////////
    public int getNodeCount() {
        return nodeMap.size();
    }
    ////////////////////////
    public int getEdgeCount() {
        return edgeMap.size();
    }
    ////////////////////////
    public int countNodesWithDegree( int degree ) {
        int sum = 0;
        for ( Node n : nodeMap.values() )
            if ( n.getDegree() == degree )
                ++sum;
        return sum;
    }
    ////////////////////////
    public int computeMaxDegree() {
        int maxDegree = 0;
        for ( Node n : nodeMap.values() ) {
            if ( maxDegree < n.getDegree() )
                maxDegree = n.getDegree();
        }
        return maxDegree;
    }
    ////////////////////////
    public String getGraphSummary() {
        StringBuffer sb = new StringBuffer();
        sb.append( "Graph Object Summary:\n" );
        sb.append( "\tNode Count: " + getNodeCount() + "\n" );
        sb.append( "\tEdge Count: "  + getEdgeCount() + "\n" );
        return sb.toString();
    }
    ////////////////////////
    public String toStringVerbose() {
        StringBuffer sb = new StringBuffer();
        sb.append( "Graph Object Dump:\n" );
        sb.append( "\tNode Count: " + getNodeCount() + "\n" );
        sb.append( "\tEdge Count: "  + getEdgeCount() + "\n" );
        sb.append( "\tNodes: \n" );
        int nodeIndex = 0;
        for ( Node n : nodeMap.values() )
            sb.append( "\t\tNode[ " + nodeIndex++ + " ]: " + n.toString() +
"\n" );
        sb.append( "\tEdges: \n" );
        int edgeIndex = 0;
        for ( Edge e : edgeMap.values() )
            sb.append( "\t\tEdge[ " + edgeIndex++ + " ]: " + e.toString() +
"\n" );
        return sb.toString();
    }
    ////////////////////////
    public static int getMaxEdgesForGraph( int nodeCount ) {
        if  ( nodeCount < 0 ) throw new IllegalArgumentException( "nodeCount
must be >= 0!" );
        if ( nodeCount == 0 ) return 0;
        int n = nodeCount - 1;
        // Use math formula sum of first n integers where n here is
nodeCount - 1
```

```
        int maxEdges = ( n * n + n )/2;
        return maxEdges;
    }
    ////////////////////////
    public boolean hasSelfLoops() {
        for ( Edge e : edgeMap.values() )
            if ( e.getN1() == e.getN2() )
                return true;
        return false;
    }
}

////////////////////////////////////////////////////////////////////////////
//

final class Node implements Comparable< Node > {

    private final String id;
    private int degree = 0;

    private Node() {
        id = null;
    }
    ////////////////////////
    public Node( String id ) {
        this.id = id;
    }
    ////////////////////////
    public String getId() {
        return id;
    }
    ////////////////////////
    public synchronized int getDegree() {
        return degree;
    }
    ////////////////////////
    public int compareTo( Node n ) {
        return getId().compareTo( n.getId() );
    }
    ////////////////////////
    public synchronized void incrementDegree() {
        ++degree;
    }
    ////////////////////////
    @Override
    public synchronized String toString() {
        return "Node: id: " + id + " degree: " + degree;
    }
}

////////////////////////////////////////////////////////////////////////////
//

final class Edge {
    private final Node n1;
    private final Node n2;
    private final String id;
    private Edge() {
        n1 = n2 = null;
        id = null;
    }
    ////////////////////////
    public Edge( Node n1, Node n2, String id ) {
        if ( n1 == null || n2== null ) throw new IllegalArgumentException(
"Nodes must not be null!" );
        if ( n1 == n2 ) throw new IllegalArgumentException( "Argument nodes
must not be the same node!" );
        this.n1 = n1;
        this.n2 = n2;
        this.id = ( id == null ) ? computeDefaultEdgeId( n1, n2 ) : id;
    }
    ////////////////////////
    public String getId() {
        return id;
    }
    ////////////////////////
    public static String computeDefaultEdgeId( Node n1, Node n2 ) {
        if ( n1 == null || n2 == null )
            throw new IllegalArgumentException( "Arguments must not be
null!" );
        if ( n1 == n2 )
            throw new IllegalArgumentException( "Argument nodes must be for
different nodes!" );
        if ( n1.compareTo( n2 ) < 0 )
            return n1.getId() + ":" + n2.getId();
        else
            return n2.getId() + ":" + n1.getId();
    }
    ////////////////////////
    public Node getN1() {
        return n1;
    }
    ////////////////////////
    public Node getN2() {
        return n2;
    }
    ////////////////////////
    @Override
    public String toString() {
        return "Edge id: " + id + " n1: " + n1.getId() + " n2: " +
n2.getId();
    }
}
```

10B

```
/* Design a class in java for implementing the graph */


import java.util.*;
import java.io.*;

public class GraphTest {
    public static void main( String [] args ) {
        test1();
        test2();
    }
    ////////////////////////
    private static void test1() {
        for ( int nodeCount = 0; nodeCount <  5; ++nodeCount ) {
            for ( int edgeCount = -1; edgeCount <= nodeCount * 10;
++edgeCount ) {
                try {
                    System.out.println( "\n----- Test case nodeCount: " +
nodeCount + " edgeCount: " + edgeCount + " -----" );
                    System.out.flush();
                    test( nodeCount, edgeCount, true );
                }
                catch ( Exception e ) {
                    System.out.println( "Graph creation failed: " +
e.getMessage() );
                }
            }
        }
        System.out.flush();
    }
    ////////////////////////
    private static void test2() {
        int nodeCount = 10000;
        int edgeCount = nodeCount * 10;
        System.out.println( "\n----- Test case nodeCount: " + nodeCount + "
edgeCount: " + edgeCount + " -----" );
        test( nodeCount, edgeCount, false );
    }
    ////////////////////////
    private static void test( int nodeCount, int edgeCount, boolean
dumpGraph ) {
        Graph rg = Graph.createRandomGraph( nodeCount, edgeCount );
        if ( !dumpGraph )
            System.out.print( rg.getGraphSummary() );
        // Dump degree historgram
        int maxDegree = rg.computeMaxDegree();
        for ( int i = 0; i  <= maxDegree; ++i ) {
            int nodeCountWithDegree = rg.countNodesWithDegree( i );
            System.out.println( "Nodes with degree " + i + ": " +
nodeCountWithDegree );
        }
        // Test for self-looping nodes
        System.out.println( "Exists self-loops: " + rg.hasSelfLoops() );
        if ( dumpGraph )
            System.out.print( rg.toStringVerbose() );
    }
}

////////////////////////////////////////////////////////////////////////////
//

final class Graph {

    private SortedMap< String, Node > nodeMap = null;
    private Map< String, Edge > edgeMap = null;

    public Graph() {
        nodeMap = new TreeMap< String, Node >( new Comparator< String >() {
            public int compare( String s1, String s2 ) {
                return s1.compareTo( s2 );
            }
        } );
        edgeMap = new HashMap< String, Edge >();
    }
    ////////////////////////
    public static Graph createRandomGraph( int nodeCount, int edgeCount ) {
        if ( nodeCount < 1 || edgeCount < 0 ) throw new
IllegalArgumentException( "nodeCount must be >= 1 and edgeCount must be >=
0!" );
        Random rnGen = new Random( System.currentTimeMillis() );
        int maxEdges = getMaxEdgesForGraph( nodeCount );
        if ( edgeCount > maxEdges )
            throw new IllegalArgumentException( "Input edgeCount (" +
edgeCount + ") exceeds maximum possible edges for graph with " + nodeCount +
" nodes!" );
        // Create empty Graph object
        Graph g = new Graph();
        // Create temp array to hold node keys - required for
getRandomEdge()
        String [] nodeKeys = new String [ nodeCount ];
        // Create and add nodeList
        for ( int i = 0; i < nodeCount; ++i ) {
            String nodeId = Integer.toString( g.getNodeCount() );
            nodeKeys[ i ] = nodeId;
            Node n = new Node( nodeId );
            g.addNode( n ); // Let list index be node's id
        }
        // Create and add edgeList
        for ( int i = 0; i < edgeCount; ++i ) {
            Edge e = Graph.getRandomEdge( rnGen, g, nodeKeys );
            g.addEdge( e );
        }
        return g;
    }
    ////////////////////////
    public void addNode( Node n ) {
        if ( n == null ) throw new IllegalArgumentException( "Argument must
be non-null!" );
        if ( nodeMap.get( n.getId() ) != null ) throw new
IllegalArgumentException( "Attempt to add node with duplicate id <" +
n.getId() + ">" );
        nodeMap.put( n.getId(), n);
    }
    ////////////////////////
    private static Edge getRandomEdge( Random rnGen, Graph g, String [] keys
) {
        if ( g.getNodeCount() < 2 ) throw new IllegalStateException(
"Attempt to add edge when < 2 nodes are in graph!" );
        if ( keys == null || keys.length != g.getNodeCount() ) throw new
IllegalArgumentException( "keys argument null or wrong size!" );
        Node n1 = null;
        Node n2 = null;
        Edge retEdge = null;
        while ( true ) {
            n1 = g.nodeMap.get( keys[ rnGen.nextInt( g.getNodeCount() ) ] );
            n2 = g.nodeMap.get( keys[ rnGen.nextInt( g.getNodeCount() ) ] );
            if ( n1 == n2 ) // Skip if already have edge between these two
nodes
                continue;
            String id = Edge.computeDefaultEdgeId( n1, n2 );
            if ( g.edgeMap.get( id ) != null )
                continue;
            retEdge = new Edge( n1, n2, id );
            break;
        }
        return retEdge;
    }
    ////////////////////////
    public void addEdge( Edge e ) {
        if ( edgeMap.get( e.getId() ) != null ) throw new
IllegalStateException( "Attemp to add edge wiith duplicate id <" + e.getId()
+ ">" );
        edgeMap.put( e.getId(), e );
        e.getN1().incrementDegree();
        e.getN2().incrementDegree();
    }
    ////////////////////////
    public int getNodeCount() {
        return nodeMap.size();
    }
    ////////////////////////
    public int getEdgeCount() {
        return edgeMap.size();
    }
    ////////////////////////
    public int countNodesWithDegree( int degree ) {
        int sum = 0;
        for ( Node n : nodeMap.values() )
            if ( n.getDegree() == degree )
                ++sum;
        return sum;
    }
    ////////////////////////
    public int computeMaxDegree() {
        int maxDegree = 0;
        for ( Node n : nodeMap.values() ) {
            if ( maxDegree < n.getDegree() )
                maxDegree = n.getDegree();
        }
        return maxDegree;
    }
    ////////////////////////
    public String getGraphSummary() {
        StringBuffer sb = new StringBuffer();
        sb.append( "Graph Object Summary:\n" );
        sb.append( "\tNode Count: " + getNodeCount() + "\n" );
        sb.append( "\tEdge Count: "  + getEdgeCount() + "\n" );
        return sb.toString();
    }
    ////////////////////////
    public String toStringVerbose() {
        StringBuffer sb = new StringBuffer();
        sb.append( "Graph Object Dump:\n" );
        sb.append( "\tNode Count: " + getNodeCount() + "\n" );
        sb.append( "\tEdge Count: "  + getEdgeCount() + "\n" );
        sb.append( "\tNodes: \n" );
        int nodeIndex = 0;
        for ( Node n : nodeMap.values() )
            sb.append( "\t\tNode[ " + nodeIndex++ + " ]: " + n.toString() +
"\n" );
        sb.append( "\tEdges: \n" );
        int edgeIndex = 0;
        for ( Edge e : edgeMap.values() )
            sb.append( "\t\tEdge[ " + edgeIndex++ + " ]: " + e.toString() +
"\n" );
        return sb.toString();
    }
    ////////////////////////
    public static int getMaxEdgesForGraph( int nodeCount ) {
        if  ( nodeCount < 0 ) throw new IllegalArgumentException( "nodeCount
must be >= 0!" );
        if ( nodeCount == 0 ) return 0;
        int n = nodeCount - 1;
        // Use math formula sum of first n integers where n here is
nodeCount - 1
```

```java
        int maxEdges = ( n * n + n )/2;
        return maxEdges;
    }
    ////////////////////////
    public boolean hasSelfLoops() {
        for ( Edge e : edgeMap.values() )
            if ( e.getN1() == e.getN2() )
                return true;
        return false;
    }
}

////////////////////////////////////////////////////////////////////////////
//

final class Node implements Comparable< Node > {

    private final String id;
    private int degree = 0;

    private Node() {
        id = null;
    }
    ////////////////////////
    public Node( String id ) {
        this.id = id;
    }
    ////////////////////////
    public String getId() {
        return id;
    }
    ////////////////////////
    public synchronized int getDegree() {
        return degree;
    }
    ////////////////////////
    public int compareTo( Node n ) {
        return getId().compareTo( n.getId() );
    }
    ////////////////////////
    public synchronized void incrementDegree() {
        ++degree;
    }
    ////////////////////////
    @Override
    public synchronized String toString() {
        return "Node: id: " + id + " degree: " + degree;
    }
}

////////////////////////////////////////////////////////////////////////////
//

final class Edge {
    private final Node n1;
    private final Node n2;
    private final String id;
    private Edge() {
        n1 = n2 = null;
        id = null;
    }
    ////////////////////////
    public Edge( Node n1, Node n2, String id ) {
        if ( n1 == null || n2== null ) throw new IllegalArgumentException(
"Nodes must not be null!" );
        if ( n1 == n2 ) throw new IllegalArgumentException( "Argument nodes
must not be the same node!" );
        this.n1 = n1;
        this.n2 = n2;
        this.id = ( id == null ) ? computeDefaultEdgeId( n1, n2 ) : id;
    }
    ////////////////////////
    public String getId() {
        return id;
    }
    ////////////////////////
    public static String computeDefaultEdgeId( Node n1, Node n2 ) {
        if ( n1 == null || n2 == null )
            throw new IllegalArgumentException( "Arguments must not be
null!" );
        if ( n1 == n2 )
            throw new IllegalArgumentException( "Argument nodes must be for
different nodes!" );
        if ( n1.compareTo( n2 ) < 0 )
            return n1.getId() + ":" + n2.getId();
        else
            return n2.getId() + ":" + n1.getId();
    }
    ////////////////////////
    public Node getN1() {
        return n1;
    }
    ////////////////////////
    public Node getN2() {
        return n2;
    }
    ////////////////////////
    @Override
    public String toString() {
        return "Edge id: " + id + " n1: " + n1.getId() + " n2: " +
n2.getId();
    }
}
```