Important Note: In this note, most of the time, we have just provided the main codes for doing a program by implementing recursion and have not declared the class or written the main() method.
In order to write the complete program, you are required to first declare a class and then write the recursive methods. An example of the code which you need to write before the methods is given below:

```
import java.io.*;
class Sample
   {
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
```

After writing the recursive methods, the main method must be written in order for the complete program to run. The main() method contains nothing but a few lines for calling the above created methods.

An example of the main method for the above program is given below:

```
public static void main()throws IOException
{
Sample ob=new Sample();
ob.display();
}
```

Fibonacci Series

Recursive Code for generating the nth term of the Fibonacci Series. This is the most widely used function for generating the Fibonacci series.

```
int fib(int n)
{
if(n<=1)
  return 0;
else if(n==2)
  return 1;
else
  return (fib(n-1)+fib(n-2));
}
```

If we want to print 1st 10 terms, then we will call the above method 10 times as follows:

```
int c;
for(int i=1;i<=10;i++)
{
c=fib(i);
System.out.print(c+"  ");
}
```

In the above code, the variable 'c' is storing the value of the 1st term, then the 2nd term, then the 3rd term till 10th terms. As soon as 'c' gets a value, we are printing it. Hence 10 terms of the Fibonacci series is generated.

Another Recursive Method	Corresponding Iterative Method

int a=0,b=1,c=0;
void fib(int i, int limit)
{
if(i<=limit)
{
c=a+b;
System.out.print(c+" ");
a=b;
b=c;
fib(i+1,limit);
}
}
 	
void display()
{
System.out.print("Enter the limit : ");
int limit=Integer.parseInt(br.readLine());
System.out.print("The Series is : "+a+" "+b+" ");
fib(3,limit);
}
	int a=0,b=1,c=0;
	void fib(int limit)
	{
	int i=3;
	while(i<=limit)
	{
	c=a+b;
	System.out.print(c+" ");
	a=b;
	b=c;
	i++;
	}
	}
	
	void display()
	{
	System.out.print("Enter the limit : ");
	int limit=Integer.parseInt(br.readLine());
	System.out.print("The Series is : "+a+" "+b+" ");
	fib(limit);
	}

The above recursive code generates and prints all the Fibonacci series term, from the 3rd term onwards. The 1st and the 2nd term are stored in the variables 'a' and 'b' respectively, while the third term which is being recursively generated is stored in the variable 'c'. Since, the 1st and the 2nd terms are known to be 0 and 1 and hence they are directly printed in the display method. This is why we are sending the starting value as 3.

Students are advised to use the 1st method whenever possible.

Generating the Fibonacci Series (ISC 2005)

```
import java.io.*;
class Recursion
{
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int a,b,c,limit;
Recursion()
{
a=0;
b=1;
c=0;
limit=0;
}
void input()throws IOException
{
System.out.print("Enter the limit : ");
limit=Integer.parseInt(br.readLine());
}
int fib(int n)
{
if(n<=1)
  return a;
else if(n==2)
  return b;
else
  return (fib(n-1)+fib(n-2));
}
void generate_fibseries()
{
System.out.println("The Fibonacci Series is:");
for(int i=1;i<=limit;i++)
{
c=fib(i);
System.out.print(c+"  ");
}
}
public static void main()throws IOException
  {
   Recursion ob=new Recursion();
   ob.input();
   ob.generate_fibseries();
  }
}
```

Note: All the functions used here are what was given in the question (ISC 2005)

Finding HCF or GCD of 2 numbers

Recursive Method 1	Recursive Method 2	Recursive Method 3	Recursive Method 4

int gcd(int p,int q)
{
if(q==0)
return p;
return gcd(q,p%q);
}
	int gcd(int p,int q)
	{
	if(p%q!=0)
	  return gcd(q,p%q);
	else
	  return q;
	}
		int gcd(int p,int q)
		{
		if(p==q)
		return p;
		else if(p>q)
		return gcd(p-q,q);
		else
		return gcd(p,q-p);
		}
			int res=1;
			int gcd(int p,int q,int i)
			{
			if(i<=p)
			{
			if(p%i==0&&q%i==0)
			res=i;
			gcd(p,q,i+1);
			}
			return res;
			}

Any of the above 4 methods can be used, for finding the GCD or HCF of any two numbers. We would advise the students to use the 1st recursive method, whenever possible.

A program on finding HCF of 2 numbers using the recursive technique came in **ISC 2006**.

Finding the LCM of 2 numbers

```
int lcm=1;
int calcLCM(int a,int b)
{
if(lcm%a==0 && lcm%b==0)
   return lcm;
 lcm++;
 calcLCM(a,b);
 return lcm;
}
void display()throws IOException
{
System.out.print("Enter the 1st number : ");
int x=Integer.parseInt(br.readLine());
System.out.print("Enter the 2nd number : ");
int y=Integer.parseInt(br.readLine());
findLCM(x,y);
System.out.println("LCM = "+lcm);
}
```

Checking for Prime Number

Another Recursive Method	Corresponding Iterative Method

int count=0;
int prime(int n,int i)
{
if(i<=n)
{
if(n%i==0)
{
count++;
}
return (prime(n,i+1));
}
else
return count;
}
void display()throws IOException
{
System.out.print("Enter the any number : ");
int n=Integer.parseInt(br.readLine());
int res=prime(n,1);
if(res==2)
System.out.println("The number is Prime");
else
System.out.println("The number is Not Prime");
}
	int count=0,i=1;
	int prime(int n)
	{
	while(i<=n)
	{
	if(n%i==0)
	{
	count++;
	}
	i++;
	}
	return count;
	}
	
	void display()throws IOException
	{
	System.out.print("Enter the any number : ");
	int n=Integer.parseInt(br.readLine());
	int res=prime(n);
	if(res==2)
	System.out.println("The number is Prime");
	else
	System.out.println("The number is Not Prime");
	}

Finding Prime Triplets

```
int count=0;
int prime(int n,int i)
{
if(i<=n)
{
   if(n%i==0)
       {
        count++;
       }
return (prime(n,i+1));
 }
else
  return count;
}
boolean isPrime(int m)
{
count=0; //before calling prime() method, count should
             be re-initialised to 0, otherwise it will keep
             on adding to the previous value of count
int res=prime(m,1);
if(res==2)
     return true;
else
      return false;
}

void display()throws IOException
{
System.out.print("Enter the start limit : ");
int x=Integer.parseInt(br.readLine());
System.out.print("Enter the end limit : ");
int y=Integer.parseInt(br.readLine());
System.out.println("The Prime Triplets are : ");
boolean a,b,c,d;
for(int i=x;i<=y;i++)
   {
   a=isPrime(i);
   b=isPrime(i+2);
   c=isPrime(i+4);
   d=isPrime(i+6);
   if(a==true && b==true && d==true)
        System.out.println(i+"\t"+(i+2)+"\t"+(i+6));
   if(a==true && c==true && d==true)
        System.out.println(i+"\t"+(i+4)+"\t"+(i+6));
   }
}
```

The consecutive prime numbers are known as **Prime Triplets** if they satisfy the following conditions:
(n, n+2, n+6) are all prime OR (n, n+4, n+6) are all prime, where n is an integer number. Here we are finding and printing all the Prime Triplets between a given range.

Note: The only recursive method used here is **prime() which** is returning the number of factors of a number. The function **isPrime()** is calling the function **prime()** and checking whether the value returned by it is 2 or not. If the value returned is 2, then the number is prime, hence it is returning true, otherwise it is returning false.

Checking for Twin prime

```
int count=0;
int prime(int n,int i)
{
if(i<=n)
{
   if(n%i==0)
       {
        count++;
       }
return (prime(n,i+1));
 }
else
  return count;
}
boolean isPrime(int m)
{
count=0;
int res=prime(m,1);
if(res==2)
return true;
else
return false;
}
void display()throws IOException
{
System.out.print("Enter the 1st number : ");
int x=Integer.parseInt(br.readLine());
System.out.print("Enter the 2nd number : ");
int y=Integer.parseInt(br.readLine());
boolean a=isPrime(x);
boolean b=isPrime(y);
if(a==true && b==true && Math.abs((x-y))==2)
 System.out.println(x+" and "+y+" are Twin primes");
else
 System.out.println("They are not Twin Primes");
}
```

Factorial of a number

Recursive Method 1

```
int factorial(int n)
 {
  if(n==0)
    return 1;
  else
    return (n*factorial(n-1));
 }
void display()throws IOException
  {
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int a=factorial(n);
System.out.print("The factorial of the number = "+a);
  }
```

Recursive Method 2

```
int f=1;
int factorial(int n)
 {
  if(n==0)
    return f;
  else
    {
     f=f*n;
     return (factorial(n-1));
    }
 }
```

The display() method will be the same as on the left.

Recursive Method 3

```
int f=1;
int factorial(int n, int i)
{
  if(i<=n)
    {
     f=f*i;
     return (factorial(n,i+1));
    }
  return f;
 }

void display()throws IOException
  {
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int a=factorial(n,0);
System.out.print("The factorial of the number = "+a);
  }
```

Corresponding Iterative Function

```
int f=1;
int factorial(int n)
 {
  int i=1;
  while(i<=n)
      {
       f=f*i;
       i++;
      }
  return f;
 }
void display()throws IOException
  {
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int a=factorial(n);
System.out.print("The factorial of the number = "+a);
  }
```

Note: The use of the 1st or 2nd method is preferred while writing recursive code for finding the factorial of any number. The 1st method is the most widely used and the shortest of all the above methods, hence students are advised to use the 1st method while finding factorial of a number using recursion.

Some Programs dealing with Factorial

1. Evaluating the Permutation Function: or P(n,r) = $\frac{n!}{(n-r)!}$

```
import java.io.*;
class Permutation
{
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int n,r;

Permutation()
 {
  n=0;
  r=0;
 }

void readInput()throws IOException
 {
  System.out.print("Enter the value of n : ");
  n=Integer.parseInt(br.readLine());
  System.out.print("Enter the value of r : ");
  r=Integer.parseInt(br.readLine());
 }

int factorial(int n)
 {
  if(n==0)
    return 1;
  else
    return (n*factorial(n-1));
 }
void compute()
 {
  int a=factorial(n);
  int b=factorial(n-r);
  int res=a/b;
  System.out.println("Answer : "+res);
  }
public static void main()throws IOException
  {
   Permutation ob=new Permutation();
   ob.readInput();
   ob.compute();
  }
}
```

2. Evaluating the Combination Function: or C(n,r) = $\frac{n!}{r!\,(n-r)!}$

```
import java.io.*;
class Combination
{
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int n,r;

Combination()
 {
  n=0;
  r=0;
 }

void readInput()throws IOException
 {
  System.out.print("Enter the value of n : ");
  n=Integer.parseInt(br.readLine());
  System.out.print("Enter the value of r : ");
  r=Integer.parseInt(br.readLine());
 }

int factorial(int n)
 {
  if(n==0)
    return 1;
  else
    return (n*factorial(n-1));
 }

void compute()
 {
  int a=factorial(n);
  int b=factorial(r);
  int c=factorial(n-r);
  int res=a/(b*c);
  System.out.println("Answer : "+res);
  }
public static void main()throws IOException
  {
   Combination ob=new Combination();
   ob.readInput();
   ob.compute();
  }
}
```

In both the above examples, we are using the recursive function (**int factorial(int n)**) for finding the factorial of a number, and we are calling this function from inside the **compute()** function to find the values of n!, r! and (n-r)! and them applying the given formula to calculate the Permutation or the Combination Function.

3. Checking whether a number is a Special Number (Krishnamurthy Number) or not

```
int factorial(int n)
 {
  if(n==0)
    return 1;
  else
    return (n*factorial(n-1));
 }
void isSpecial(int n)
 {
 int d,s=0,copy=n;
 while(n>0)
   {
    d=n%10;
    s=s+factorial(d);
    n=n/10;
   }
 if(s==copy)
  System.out.println("The number is Special");
 else
   System.out.println("The number is Not Special");
 }
void display()throws IOException
  {
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
isSpecial(n);
  }
```

Note: In such programs, you may be asked to take the variable 'n' as instance variables. In that case you don't need to take it as parameter to the function **isSpecial()**.

It can also be given that you create a separate function for taking inputs from the user to the variable 'n'. In that case you create that separate function, and write the code for inputting 'n' inside it. If you do this, then you don't need to input 'n' in the **display()** method.

4. Finding the series : $S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$ (x and n are inputs from the user)

```
int factorial(int n)
 {
  if(n==0)
    return 1;
  else
    return (n*factorial(n-1));
 }
void calcSeries(int x, int n)
 {
 double s=0.0;
 for(int i=0;i<n;i++)
 {
  num=(int)Math.pow(x,i);
  den=factorial(i);
  s=s+(num/den);
 }
 System.out.println("Output : "+s);
 }
```

```
void display()throws IOException
  {
   System.out.print("Enter the value of n : ");
   int n=Integer.parseInt(br.readLine());
   System.out.print("Enter the value of x : ");
   int x=Integer.parseInt(br.readLine());
   calcSeries(x,n);
  }
```

Note: In such programs, you may be asked to take the variable 'x' and 'n' as instance variables. Then, you will need to define them just after declaring the class. In that case you don't need to take them as parameters to the function **calcSeries()**.

It you are asked to create a separate function for taking inputs from the user to the variables 'x' and 'n', then you don't need to input them in the **display()** method.

For the series : $S = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \cdots + \frac{x^n}{n!}$

```
void calcSeries(int x, int n)
 {
 double s=0.0;
 for(int i=1;i<=n;i++)
 {
  num=(int)Math.pow(x,i);
  den=factorial(i);
  if(x%2==0)
    s=s-(num/den);
  else
    s=s+(num/den);
 }
 System.out.println("Output : "+s);
}
```

For the series : $S = 1 + \frac{x^2}{3!} + \frac{x^3}{4!} + \frac{x^4}{5!} + \cdots + \frac{x^n}{(n+1)!}$

```
void calcSeries(int x, int n)
 {
  double s=1.0;
  for(int i=2;i<=n;i++)
  {
   num=(int)Math.pow(x,i);
   den=factorial(i+1);
   s=s+(num/den);
  }
  System.out.println("Output : "+s);
}
```

Extracting Digits from a number and performing any given operation

Operation from First digit onwards	Operation from Last digit onwards

void numOp(int n)
{
if(n>0)
{
int d=n%10;
numOp(n/10);
Write the operation you want to perform with the digits coming out from the beginning over here.
}
}
	void numOp(int n)
	{
	if(n>0)
	{
	int d=n%10;
	Write the operation you want to perform with the digits coming out from the end over here.
	numOp(n/10);
	}
	}

In the 1st code, we are taking out the digits from the end of the number, and without performing any operation with the digits extracted from the last, we are again calling the recursive function with the number reduced by 10 times.
We then write the operation we want to perform after the line which is calling the function recursively, because by doing so, we would be using the LIFO property of the stack used in recursion. Digits extracted from the end, were placed in the beginning of the stack, hence, when we pop out the digits, we would be getting the First digit of the number first, then the second and so on. Thus in this case we get digits from the beginning of a number.

In the 2nd code above code, we are taking out the digits from the end of the number, and are performing the operation with the digits extracted before calling the recursive function with the number reduced by 10 times. Thus in this case we get digits from the end of a number.

Some Programs dealing with digits of a number

1. Finding the sum of the digits of a number

Recursive Method 1	Recursive Method 2

int s=0;
int sumDig(int n)
{
if(n>0)
{
int d=n%10;
sumDig(n/10);
s=s+d;
}
return s;
}
 	
void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int x=sumDig(n);
System.out.println("Sum of digits = "+x);
}
	int s=0;
	int sumDig(int n)
	{
	if(n>0)
	{
	int d=n%10;
	s=s+d;
	sumDig(n/10);
	}
	return s;
	}
	
	In the 1st method we are adding digits from the beginning, while in the 2nd method, we are adding the digits from the end.
	
	The difference between the 2 method is of shifting the line **s=s+d;** which in the 1st method is written after the recursive call, while in the 2nd is written before the recursive call.

Important Note:

For finding the sum of the **square of the digits**, write the above code. Just change the line **s=s+d;** into **s=s+d*d;** or **s=s+(int)Math.pow(d,2);**

For finding the sum of the **cube of the digits**, write the above code. Just change the line **s=s+d;** into **s=s+d*d*d;** or **s=s+(int)Math.pow(d,3);**

Another Important Recursive Method (Method 3)

Use this method, when you are not provided with or are asked not to take any separate variable for storing the sum. In the above 2 examples we used a variable 's' as an instance variable for storing sum. in the below given example, we are not using any variable for storing the sum.

```
int sumDig(int n)
 {
  if(n==0)
   return 0;
  else
   {
    int d=n%10;
    return (d+sumDig(n/10));
   }
 }
```

For finding the sum of the **square of the digits**, just change the line **return (d+sumDig(n/10));** into **return (d*d+sumDig(n/10));**

For finding the sum of the **cube of the digits**, just change the line **return (d+sumDig(n/10));** into **return (d*d*d+sumDig(n/10));**

2. Finding whether a number is a Magic Number or not (ISC 2009)

```
import java.io.*;
class Magic
{
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int n;
Magic()
  {
   n=0;
  }
void getnum()throws IOException
 {
   System.out.print("Enter any number: ");
   n=Integer.parseInt(br.readLine());
  }
int sumDig(int x)
 {
  if(x==0)
    return 0;
  else
   {
    int d=x%10;
    return (d+ sumDig(x/10));
   }
 }
void isMagic()
{
int a=n;
while(a>9)
{
a=sumDig(a);
}
if(a==1)
   System.out.print(n+" is a Magic Number");
else
   System.out.print(n+" is Not a Magic Number");
}
public static void main()throws IOException
  {
   Magic ob=new Magic();
   ob.getnum(b);
   ob.isMagic();
  }
}
```

Note: A Magic Number is a number whose eventual sum of the digits is equal to 1. This addition of digits is performed again and again till the number itself becomes a single digit number. Example, 28

In this program we have used the 3rd method of finding the sum of the digits.

Important Note: In such programs you may be asked to input the number in a separate function like we have used above. We have taken the value of 'n' as input from the user in the function **getnum()**.

You can also be asked to initialize the variable 'n' inside a function with some parameter passed to that function. In such a case you don't input 'n' in that function, but you input it inside main() and pass this input to that initializing function.

Eg. In the above program you can write the getnum() method as:

```
void getnum(int num)
 {
   n=num;
  }
```

Then the main() method will be:

```
public static void main()throws IOException
  {
   Magic ob=new Magic();
   System.out.print("Enter any number: ");
   int b=Integer.parseInt(br.readLine());
   ob.getnum(b);
   ob.isMagic();
  }
```

3. Finding whether a number is a Happy Number or not (ISC 2012)

```
import java.io.*;
class Happy
{
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int n;

Happy()
 {
  n=0;
 }

void getnum(int nn)
 {
  n=nn;
 }

int sum_sq_digits(int x)
 {
  if(x==0)
    return 0;
  else
   {
    int d=x%10;
    return (d*d+ sum_sq_digits(x/10));
   }
 }

void ishappy()
{
int a=n;
while(a>9)
{
a=sum_sq_digits(a);
}
if(a==1)
   System.out.print(n+" is a Happy Number");
else
   System.out.print(n+" is Not a Happy Number");
}

public static void main()throws IOException
  {
   Happy ob=new Happy();
   System.out.print("Enter any number: ");
   int b=Integer.parseInt(br.readLine());
   ob.getnum(b);
   ob.ishappy();
  }
}
```

Note: In this program we have used the 3rd method of finding the sum of the square of the digits.
All the functions used here are what was given in the question (Question 10 of ISC 2012)

4. Finding whether a number is an Armstrong Number or not

```
import java.io.*;
class Armstrong
{
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int n;

Armstrong()
  {
   n=0;
  }
void readNum()throws IOException
 {
   System.out.print("Enter any number: ");
   n=Integer.parseInt(br.readLine());
  }
int sumCubeDig(int x)
 {
  if(x==0)
    return 0;
  else
   {
    int d=x%10;
    return (d*d*d+ sumCubeDig(x/10));
   }
 }

void isArm()
{
int a=sumCubeDig(n);
if(a==n)
   System.out.print(n+" is an Armstrong Number");
else
   System.out.print(n+" is Not an Armstrong Number");
}

public static void main()throws IOException
  {
   Armstrong ob=new Armstrong();
   ob.readNum();
   ob.isArm();
  }
}
```

Note: An Armstrong Number is a number whose sum of the cube of the digits is equal to the original number. Example, $153 = 1^3+5^3+3^3$

In this program we have used the 3rd method of finding the sum of the cube of the digits.

5. Finding frequency of the digits of a number

```
int A[]={0,0,0,0,0,0,0,0,0,0};
void count(int n)
{
  if(n>0)
   {
    int d=n%10;
    A[d]++;
    count(n/10);
    }
}
void display()throws IOException
{
   System.out.print("Enter any number : ");
   int n=Integer.parseInt(br.readLine());
   System.out.println("Digit\t\tFrequency");
   count(n);
   for(int i=0;i<10;i++)
   {
    if(A[i]!=0)
       System.out.println(i+"\t\t"+A[i]);
   }
 }
```

Note: In this program, we have taken an integer Array of size=10 for counting the frequency of each digit present in a number. This array has to be declared as an instance variable. Each cell of the array has been initialized to zero.

Cell with index 0 will store the frequency of digit 0, cell with index 1 will store the frequency of digit 1 and so on.

In the recursive function, we are extracting the digits one by one, and incrementing the corresponding cell of the array.

So after the recursive function ends, we have in the array the frequency of each digit. In the display() method, we are printing only those cells of the array whose value is not zero.

The "\t" is for giving tab spaces.

6. Printing the digits of a number in words (ISC 2007)

```
import java.io.*;
class Convert
{
static BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
int n;
Convert()
  {
  n=0;
  }
void inpnum()throws IOException
 {
  System.out.print("Enter any number: ");
  n=Integer.parseInt(br.readLine());
  extdigit(n);
  }
void extdigit(int n)
 {
  if(n>0)
   {
    int d=n%10;
    extdigit(n/10);
    num_to_words(d);
    }
}
void num_to_words(int x)
{
  switch(x)
  {
   case 0: System.out.print("Zero "); break;
   case 1: System.out.print("One "); break;
   case 2: System.out.print("Two "); break;
   case 3: System.out.print("Three "); break;
   case 4: System.out.print("Four "); break;
   case 5: System.out.print("Five "); break;
   case 6: System.out.print("Six "); break;
   case 7: System.out.print("Seven "); break;
   case 8: System.out.print("Eight "); break;
   case 9: System.out.print("Nine "); break;
  }
}
public static void main()throws IOException
  {
   Convert ob=new Convert();
   System.out.print("Output : ");
   ob.inpnum();
  }
}
```

Note: All the functions used here are what was given in the question (ISC 2007)

In the above program, we are inputting the number in the **inpnum()** method. After inputting, we are sending this number to the function **extdigit(int n)** which is extracting one digit at a time. Here we have used the 1st recursive method of extracting digits from the beginning. As soon as we are getting the digits from the stack using the LIFO property, we are sending it to the **num_to_words(int x)** method, which is printing the digit in words.

Reversing a Number

Recursive Method 1 Without Return Type	Recursive Method 2 With Return type	Corresponding Iterative Code

void rev(int n)
{
if(n>0)
{
int d=n%10;
System.out.print(d);
rev(n/10);
}
}
	int r=0;
	int rev(int n)
	{
	if(n>0)
	{
	int d=n%10;
	r=r*10+d;
	return (rev(n/10));
	}
	else
	return r;
	}
		int r=0;
		int rev(int n)
		{
		while(n>0)
		{
		int d=n%10;
		r=r*10+d;
		n=n/10;
		}
		return r;
		}

void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
System.out.print("Reverse of the number = ");
rev(n);
}
	void display()throws IOException
	{
	System.out.print("Enter any number : ");
	int n=Integer.parseInt(br.readLine());
	int x=rev(n);
	System.out.print("Reverse of the number = "+x);
	}	

Checking for Palindrome Number

```
int r=0;
int rev(int n)
{
 if(n>0)
  {
   int d=n%10;
   r=r*10+d;
   return (rev(n/10));
   }
 else
   return r;
}
```

```
void display()throws IOException
 {
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int x=rev(n);
if(x==n)
System.out.println("The number is Palindrome");
else
System.out.println("The number is Not Palindrome");
}
```

Finding power of a number (a^b)

Recursive Method 1	Recursive Method 2	Corresponding Iterative Code

int power(int a, int b)
{
if(b==0)
return 1;
else
return (a*power(a,b-1));
}
 	
void display()throws IOException
{
System.out.print("Enter any number : ");
int m=Integer.parseInt(br.readLine());
System.out.print("Enter the power : ");
int n=Integer.parseInt(br.readLine());
int x=power(m,n);
System.out.print("Result = "+x);
}
	int p=1;
	int power(int a, int b)
	{
	if(b==0)
	return p;
	else
	{
	p=p*a;
	return (power(a,b-1));
	}
	}
	
	The display() method will be the same as on the left.
		int p=1;
		int power(int a, int b)
		{
		int i=1;
		while(i<=b)
		{
		p=p*a;
		i++;
		}
		return p;
		}
		
		The display() method will be the same as on the left.

Extracting Numbers from a limit upto another limit

1. Printing all the Even Numbers starting from 'p' till 'q'

Recursive Method	Corresponding Iterative Function

void even(int p, int q)
{
if(p<=q)
{
if(p%2==0)
System.out.println(p);
even(p+1,q);
}
}
void display()throws IOException
{
System.out.print("Enter the lower limit : ");
int p=Integer.parseInt(br.readLine());
System.out.print("Enter the upper limit : ");
int q=Integer.parseInt(br.readLine());
System.out.println("The even numbers are :");
even(p,q);
}
	void even(int p, int q)
	{
	while(p<=q)
	{
	if(p%2==0)
	System.out.println(p);
	p++;
	}
	}
	void display()throws IOException
	{
	System.out.print("Enter the lower limit : ");
	int p=Integer.parseInt(br.readLine());
	System.out.print("Enter the upper limit : ");
	int q=Integer.parseInt(br.readLine());
	System.out.println("The even numbers are :");
	even(p,q);
	}

One by one, the numbers are coming in the variable 'p', and you need to write what you want to do with that number. In the above example, we are printing all those numbers from p to q which are even.

2. Finding Sum of all the Even and Odd Numbers separately from 'p' till 'q'

```
int so=0,se=0;
void sumOE(int p, int q)
{
  if(p<=q)
    {
      if(p%2==0)
          se=se+p;
      else
          so=so+p;
      sumOE(p+1,q);
    }
 }
void display()throws IOException
  {
System.out.print("Enter the lower limit : ");
int p=Integer.parseInt(br.readLine());
System.out.print("Enter the upper limit : ");
int q=Integer.parseInt(br.readLine());
sumOE(p,q);
System.out.println("Sum of even numbers = "+se);
System.out.println("Sum of odd numbers = "+so);
  }
```

3. Finding Sum of all the Even and Odd Numbers separately from 'p' till 'q'

```
int so=0,se=0;
void sumOE(int p, int q)
{
  if(p<=q)
    {
      if(p%2==0)
          se=se+p;
      else
          so=so+p;
      sumOE(p+1,q);
    }
 }
void display()throws IOException
  {
System.out.print("Enter the lower limit : ");
int p=Integer.parseInt(br.readLine());
System.out.print("Enter the upper limit : ");
int q=Integer.parseInt(br.readLine());
sumOE(p,q);
System.out.println("Sum of even numbers = "+se);
System.out.println("Sum of odd numbers = "+so);
  }
```

Using this technique, you can also check for all the prime numbers or any other number within a given range. Just pass on the value of 'p' to a function which is checking whether that number is prime or not or to any appropriate function.

4. Finding the factors of a number

```
void factors(int n, int i)
{
  if(i<=n)
    {
     if(n%i==0)
         System.out.print(i+"  ");
     factors(n,i+1);
    }
 }
```

```
void display()throws IOException
  {
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
System.out.print("Factors of the number are : ");
factors(n,1);
  }
```

5. Checking for Perfect Number

```
int sum=0;
int factors(int n, int i)
{
  if(i<n)
    {
     if(n%i==0)
         sum=sum+i;
     factors(n,i+1);
    }
 }
```

```
void display()throws IOException
  {
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int f=factors(n,1);
if(f==n)
System.out.println("The Number is Perfect");
else
System.out.println("The Number is Not Perfect");
  }
```

Finding the Prime Factors of a number

Recursive Method	Corresponding Iterative Method

void primeFact(int n,int i)
{
if(n>1)
{
if(n%i == 0)
{
System.out.print(i+" ");
primeFact(n/i,i);
}
else
primeFact(n,i+1);
}
}
void display()throws IOException
{
System.out.print("Enter the any number : ");
int n=Integer.parseInt(br.readLine());
System.out.print("Prime Factors of the number : ");
primeFact(n,2);
}
	int primeFact(int n)
	{
	int i=2;
	while(n>1)
	{
	if(n%i == 0)
	{
	System.out.print(i+" ");
	n=n/i;
	}
	else
	i++;
	}}
	void display()throws IOException
	{
	System.out.print("Enter the any number : ");
	int n=Integer.parseInt(br.readLine());
	System.out.print("Prime Factors of the number : ");
	primeFact(n);
	}

1. Finding Sum of the Prime Factors of a number

```
int sum=0;
int primeFact(int n,int i)
{
if(n>1)
{
   if(n%i == 0)
       {
        sum=sum+i;
        return (primeFact(n/i,i));
       }
   else
        return (primeFact(n,i+1));
}
else
 return sum;
}
void display()throws IOException
{
System.out.print("Enter the any number : ");
int n=Integer.parseInt(br.readLine());
System.out.print("Prime Factors of the number : ");
primeFact(n,2);
}
```

2. Checking for Smith Number

```
int sum=0;
int primeFact(int n,int i)
{
if(n>1)
{
   if(n%i == 0)
       {
          sum=sum+sumDig(i);
          return (primeFact(n/i,i));
       }
   else
        return (primeFact(n,i+1));
}
else
return sum;
}

int sumDig(int n)
 {
  if(n==0)
   return 0;
  else
   {
    int d=n%10;
    return (d+sumDig(n/10));
   }
 }
void display()throws IOException
{
System.out.print("Enter the any number : ");
int n=Integer.parseInt(br.readLine());
int sd=sumDig(n);
int sf=primeFact(n,2);
if(sd==sf)
System.out.println("The Number is a Smith Number");
else
System.out.println("The Number is Not a Smith
Number");
}
```

Note: The recursive method **sumDig()** is returning us the sum of the digits of a number, while the recursive method **primeFact()** is returning us the sum of the prime factors of a number. The method **primeFact()** is the same we used for finding the sum of the prime factors above, with the only addition being that we are first sending that prime factor to the **sumDig()** function and then adding it to the sum. This is done to ensure that we meet with the condition of checking for a Smith Number and hence get the sum of the digits of all those prime factors which are more than one digit.

Conversion Between Number Systems

1. Decimal to Binary Conversion

<table>
<tr><th>Recursive Method 1</th><th colspan="2">Recursive Method 2</th><th rowspan="2">Corresponding Iterative Code</th></tr>
<tr><th>Without Return Type</th><th colspan="2">With Return type</th></tr>
<tr><td>void binary(int n)
{
if(n>0)
{
int d=n%2;
binary(n/2);
System.out.print(d);
}
}</td><td>int bin=0;
int binary(int n)
{
if(n>0)
{
int d=n%2;
binary(n/2);
bin=bin*10+d;
}
return bin;
}</td><td>int bin=0,c=0;
int binary(int n)
{
if(n>0)
{
int d=n%2;
bin=bin+d*(int)Math.
pow(10,c++);
binary(n/2);
}
return bin;
}</td><td>int bin=0,c=0;
int binary(int n)
{
while(n>0)
{
int d=n%2;
bin=bin+d*(int)Math.po
w(10,c++);
n=n/2;
}
return bin;
}</td></tr>
<tr><td>void display()throws
IOException
{
System.out.print("Enter any
number : ");
int
n=Integer.parseInt(br.readLine());
System.out.print("Binary = ");
binary(n);
}</td><td colspan="3">void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int x=binary(n);
System.out.println("Binary = "+x);
}</td></tr>
</table>

2. Binary to Decimal Conversion

Recursive Method	Corresponding Iterative Method

int dec=0,c=0;
int decimal(long n)
{
if(n>0)
{
int d=(int)n%10;
dec=dec+d*(int)Math.pow(2,c++);
decimal(n/10);
}
return dec;
}
 	
void display()throws IOException
{
System.out.print("Enter any number : ");
long n=Long.parseLong(br.readLine());
long x=decimal(n);
System.out.println("Decimal Equivalent = "+x);
}
	int dec=0,c=0;
	int decimal(long n)
	{
	while(n>0)
	{
	int d=(int)n%10;
	dec=dec+d*(int)Math.pow(2,c++);
	n=n/10;
	}
	return dec;
	}
	
	void display()throws IOException
	{
	System.out.print("Enter any number : ");
	long n=Long.parseLong(br.readLine());
	long x=decimal(n);
	System.out.println("Decimal Equivalent = "+x);
	}

3. Decimal to Octal Conversion (ISC 2011)

<table>
<tr><th>Recursive Method 1</th><th colspan="2">Recursive Method 2</th><th rowspan="2">Corresponding Iterative Code</th></tr>
<tr><th>Without Return Type</th><th colspan="2">With Return type</th></tr>
<tr><td>void octal(int n)
{
if(n>0)
{
int d=n%8;
octal(n/8);
System.out.print(d);
}
}</td><td>int oct=0;
int octal(int n)
{
if(n>0)
{
int d=n%8;
octal(n/8);
oct=oct*10+d;
}
return oct;
}</td><td>int oct=0,c=0;
int octal(int n)
{
if(n>0)
{
int d=n%8;
oct=oct+d*(int)Math.
pow(10,c++);
octal(n/8);
}
return oct;
}</td><td>int oct=0,c=0;
int octal(int n)
{
while(n>0)
{
int d=n%2;
oct=oct+d*(int)Math.po
w(10,c++);
n=n/2;
}
return oct;
}</td></tr>
<tr><td>void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
System.out.print("Octal = ");
octal (n);
}</td><td colspan="3">void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int x=octal(n);
System.out.println("Octal = "+x);
}</td></tr>
</table>

Note: The above recursive methods having return types, and parameters can also be written without return types and parameters. In such a case, you need to take the decimal number 'n' as an instance variable. Example:

```
int oct=0;
void octal()
{
if(n>0)
{
int d=n%8;
n=n/8;
octal();
oct=oct*10+d;
}
}
```

The you can print the value of the variable 'oct' inside any function.

4. Octal to Decimal Conversion

Recursive Method	Corresponding Iterative Method

int dec=0,c=0;
int decimal(int n)
{
if(n>0)
{
int d=n%10;
dec=dec+d*(int)Math.pow(8,c++);
decimal(n/10);
}
return dec;
}
 	
void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int x=decimal(n);
System.out.println("Decimal Equivalent = "+x);
}
	int dec=0,c=0;
	int decimal(int n)
	{
	while(n>0)
	{
	int d=n%10;
	dec=dec+d*(int)Math.pow(8,c++);
	n=n/10;
	}
	return dec;
	}
	
	void display()throws IOException
	{
	System.out.print("Enter any number : ");
	int n=Integer.parseInt(br.readLine());
	int x=decimal(n);
	System.out.println("Decimal Equivalent = "+x);
	}

5. Decimal to Hexadecimal Conversion

<table>
<tr><th>Recursive Method 1</th><th>Recursive Method 2</th><th rowspan="2">Corresponding Iterative Code</th></tr>
<tr><th>Without Return Type</th><th>With Return type</th></tr>
<tr><td>char a[]={'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
void hexa(int n)
{
if(n>0)
{
int d=n%16;
hexa(n/16);
System.out.print(a[d]);
}
}</td><td>char a[]={'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
String h="";
String hexa(int n)
{
if(n>0)
{
int d=n%16;
hexa(n/16);
h=h+a[d];
}
return h;
}</td><td>char a[]={'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
String h="";
String hexa(int n)
{
while(n>0)
{
int d=n%16;
h=a[d]+h;
n=n/16;
}
return h;
}</td></tr>
<tr><td>void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
int x=decimal(n);
System.out.println("Decimal Equivalent = "+x);
}</td><td colspan="2">void display()throws IOException
{
System.out.print("Enter any number : ");
int n=Integer.parseInt(br.readLine());
String x=hexa(n);
System.out.println("HexaDecimal Equivalent = "+x);
}</td></tr>
</table>

In the above program of converting a Decimal number into it's equivalent Hexadecimal number, we have taken the use of a character array in which we have stored the basic digits of Hexadecimal Number system.

When we are extracting a digit from the decimal number, we are taking out its Hexadecimal equivalent from the Array and adding it to the new String. In this way, if we get 5 in the variable 'd', then a[5] will give us '5' which will be added to the String 'h'. If we get a number greater than 9, like 12, then a[12] will give us 'C' which will be added to the String 'h'. Thus finally we will have the Hexadecimal equivalent of a Decimal number.

We are making use of the LIFO property of stacks used in recursion, that is why we will be getting digits from the beginning, even though we are extracting digits from the end.

6. Hexadecimal to Decimal Conversion

Recursive Method	Corresponding Iterative Method

char a[]={'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
int dec=0,c=0;
int decimal(String n,int i)
{
if(i>=0)
{
char ch=n.charAt(i);
ch=Character.toUpperCase(ch);
for(int k=0;k<16;k++)
{
if(a[k]==ch)
{
dec=dec+k*(int)Math.pow(16,c++);
break;
}
}
decimal(n,i-1);
}
return dec;
}
 	
void display()throws IOException
{
System.out.print("Enter any number : ");
String n=br.readLine();
int len=n.length();
int x=decimal(n,len-1);
System.out.println("Decimal Equivalent = "+x);
}
	char a[]={'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
	int dec=0,c=0;
	int decimal(String n,int i)
	{
	if(i>=0)
	{
	char ch=n.charAt(i);
	ch=Character.toUpperCase(ch);
	for(int k=0;k<16;k++)
	{
	if(a[k]==ch)
	{
	dec=dec+k*(int)Math.pow(16,c++);
	break;
	}
	}
	decimal(n,i-1);
	}
	return dec;
	}
	
	void display()throws IOException
	{
	System.out.print("Enter any number : ");
	String n=br.readLine();
	int len=n.length();
	int x=decimal(n,len-1);
	System.out.println("Decimal Equivalent = "+x);
	}